Chapter 4

Statistical Modeling

4.1 PDF Modeling Introduction and Notation

The probability density function (PDF) of a random variable (RV) X is defined by

$$p(x) \triangleq \lim_{\delta \to 0} \frac{\Pr\{x - \delta/2 < X < x + \delta/2\}}{\delta}.$$

It is called a *density* function because it is the ratio of probability mass to differential area (or volume). Note that a particular value of X is written in lower case x. The PDF $p(x)$ is regarded as a function of the particular value x. When a different RV is used, for example $p(z)$, the meaning of the function $p(\ )$ changes to that defined for the RV Z. When necessary, for example in the expression $p(T(x))$, we use a subscript. For example, when $z = T(x)$, we would write $p_z(T(x))$ to make it clear that $p(\ )$ is the PDF of RV Z. For multi-dimensional vectors written in bold notation, for example $\mathbf{x} \in \mathcal{R}^P$, the meaning of the density is extended to a density with respect to a differential volume in the P-dimensional space.

The simplest way to estimate the probability density of data is by histogram. A histogram is obtained by dividing the space of the RV into "bins", then counting the number of occurrences of the training data in each bin. A second step of smoothing or curve-fitting can be used to avoid the effects of random error. A method of PDF estimation that has become popular is that of Gaussian mixtures (GM). This can be regarded as the process of curve-fitting to a histogram where the curve is constrained to be a sum of positive Gaussian-shaped functions (*modes* or *kernels*), each with a different mean and variance. It also has the statistical interpretation of a mixture density - where each sample of the RV is regarded as having been a member of a sub-class corresponding to each mode. We will devote Section 4.3 to GM PDF estimation.

Multidimensional data, $\mathbf{x} \in R^P$, can be modeled by a multidimensional GM. However, when data consists of K samples of dimension P, it is not necessary or even desirable to group all the data together into a single $K \times P$-dimensional sample. In the simplest case, all K samples are independent and we may regard them as samples of the same RV. Normally, however, they are not independent. The Markovian principle assumes consecutive samples are statistically independent when conditioned on knowing the samples that preceded it. This leads to an elegant solution, the hidden Markov model (HMM), which employs a set of M PDFs of dimension P. The HMM regards each of the K samples as having originated from one of M possible states and there is a distinct probability that the underlying model "jumps" from one state to another. We discuss the HMM, which uses GM to model each state PDFs, in section 4.4.

We discuss additional PDF models in the last section.

4.2 When is a PDF estimate good?

4.3 PDF Estimation using Gaussian Mixtures

This section is concerned with the general PDF estimation problem. Let $p(\mathbf{z})$ be the PDF of $\mathbf{z}$ which must be estimated from training samples. If $p(\mathbf{z})$ is continuous, it may be approximated to arbitrary accuracy by any kernel-based estimator [35], such as the method of Gaussian Mixtures (GM) [36] given enough terms.

4.3.1 Gaussian Mixtures

The GM form of the PDF for $\mathbf{z} \in \mathcal{R}^P$ is given by

$$p(\mathbf{z}) = \sum_{i=1}^{L} \alpha_i \, \mathcal{N}(\mathbf{z}, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \tag{4.1}$$

where

$$\mathcal{N}(\mathbf{z}, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) = (2\pi)^{-P/2} \, |\boldsymbol{\Sigma}_i|^{-1/2} \exp\left\{ -\frac{1}{2} (\mathbf{z} - \boldsymbol{\mu}_i)' \; \boldsymbol{\Sigma}_i^{-1} \; (\mathbf{z} - \boldsymbol{\mu}_i) \right\}.$$

The L mixture components are called *modes*. The GM parameters are denoted $\Lambda = \{\alpha_i, \; \boldsymbol{\mu}_i, \; \boldsymbol{\Sigma}_i\}$. The most commonly used method for finding the maximum likelihood estimate of the parameters from a training set is the E-M algorithm [36], [37].

4.3.2 Gaussian Mixtures and the E-M Algorithm

The EM algorithm is an effecive way to perform maximum likelihood (ML) estimation when the data PDF can be easily maximized if a certain set of unknown parameters are known. These "unknown" parameters, or missing data, are the mode assignments. The mode assignments can be understood if we assume that each data sample from the Gaussian mixture had been produced by exactly one of the modes. The mode assignment for sample n is denoted k_n and $\mathbf{k}$ denotes a particular set of assignments $\mathbf{k} = \{k_1, k_2 \ldots k_N\}$.

Derivation of the EM Algorithm for GM

Let $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2 \ldots \mathbf{x}_K\}$ be a collection of data. The Q-function is defined as the expected "complete" log-PDF where the expectation is carried out over the conditional distribution of the "missing data", given $\mathbf{X}$, using the current best estimate of the PDF parameters Λ, and the log-PDF is written in terms of the new values of the parameters to be estimated, Λ':

$$Q(\Lambda, \Lambda') \triangleq \mathrm{E}_{\mathbf{k}|\mathbf{X};\Lambda}\{\log \; p(\mathbf{X}, \mathbf{k}; \Lambda')\} = \sum_{\mathbf{k}} \; p(\mathbf{k}|\mathbf{X}; \Lambda) \log \; p(\mathbf{X}, \mathbf{k}; \Lambda').$$

Expanding,

$$\begin{aligned} Q(\Lambda, \Lambda') &= \textstyle\sum_{\mathbf{k}} \; p(\mathbf{k}|\mathbf{X}; \Lambda) \; \sum_n \; \log \; p(\mathbf{x}_n, k_n; \Lambda') \\ &= \textstyle\sum_n \; \sum_{\mathbf{k}} \; p(\mathbf{k}|\mathbf{X}; \Lambda) \; \log \; p(\mathbf{x}_n, k_n; \Lambda') \\ &= \textstyle\sum_n \; \sum_{k_n} \; \sum_{\mathbf{k}_{\bar{n}}} \; p(\mathbf{k}_n, \mathbf{k}_{\bar{n}}|\mathbf{X}; \Lambda) \; \log \; p(\mathbf{x}_n, k_n; \Lambda'), \end{aligned}$$

where $\mathbf{k}_{\bar{n}}$ are are the assignments not associated with sample n. The inner summation is a marginalization

$$\sum_{\mathbf{k}_{\bar{n}}} \; p(k_n, \mathbf{k}_{\bar{n}}|\mathbf{X}; \Lambda) = p(k_n|\mathbf{X}; \Lambda).$$

Thus,

$$\begin{aligned} Q(\Lambda, \Lambda') &= \textstyle\sum_n \; \sum_{k_n} \; p(k_n|\mathbf{X}; \Lambda) \; \log \; p(\mathbf{x}_n, k_n; \Lambda') \\ &= \textstyle\sum_n \; \sum_{k_n} \; p(k_n|\mathbf{x}_n; \Lambda) \; \log \; p(\mathbf{x}_n, k_n; \Lambda') \\ &= \textstyle\sum_n \; \sum_{k_n} \; \omega_{k_n,n} \log \; p(\mathbf{x}_n, k_n; \Lambda'), \end{aligned}$$

where the conditional model probabilities $\omega_{i,n}$ are defined as

$$\omega_{i,n} \triangleq p(i|\mathbf{x}_n) = \frac{p(i, \mathbf{x}_n)}{p(\mathbf{x}_n)} = \frac{p(\mathbf{x}_n|i) \; p(i)}{\sum_j p(\mathbf{x}_n|j) \; p(j)} = \frac{\mathcal{N}(\mathbf{x}_n, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \; \alpha_i}{\sum_j \mathcal{N}(\mathbf{x}_n, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \; \alpha_j}.$$

The maximization of Λ' can be carried out on the quantity

$$L(\Lambda') = \sum_n \; \sum_i \; \gamma_n \; \omega_{i,n} \log \; p(\mathbf{x}_n, i; \Lambda'),$$

where we have added *data weights*, γ_n, which define a probabilistic weights for each data sample. This could be interpreted as adjusting the influence of a training sample as though sample n was replicated γ_n times, or can be thought of as the probabilistic certainty that sample n is indeed valid. By collecting γ_n and $\omega_{i,n}$ together into a quantity $w_{i,n}$, we have

$$L(\Lambda') = \sum_n \sum_i w_{i,n} \log\ p(\mathbf{x}_n, i; \Lambda'), \tag{4.2}$$

where

$$w_{i,n} = \gamma_n\ \omega_{i,n}.$$

The algorithm in Table 4.1, maximizes (4.2) over Λ' at each iteration. While correct, is representative only. Actual computation requires careful attention to numerical issues which are discussed below.

4.3.3 Implementation Overview

In the sections that follow, we discuss the subtleties associated with practical implementations of the E-M algorithm. We also discribe a complete MATLAB library for training, evaluating, and visualizing PDF's of high dimensions. The Gaussian mixture parameters are organized into a structure. The GPARM structure for feature dimension DIM with NMODES modes has the form shown in Table 4.2. To illustrate the use of the structure in MATLAB, if `gparm` is the name of the Gaussian mixture parameters, then the mixing weight of the third mode is accessed as `gparm.modes(3).weight`. A vector containing all the weights is created as follows: `wts = [gparm.modes.weight]'`, whereupon `wts` is a NMODE-by-1 vector of mixing weights. The meaning of each parameter in the structure will be described. The correspondence between the mathematical symbols and the MATLAB variables are tabulated in Table 4.3. Some of these symbols are already defined. The rest will be defined later. The E-M algorithm of Table 4.1 is implemented by subroutine `gmix_step.m`. Training can be accomplished by calling `gmix_step.m` repeatedly. There are, however, subtleties having to do with how the GM is initialized and how the number of modes is chosen. Modes can be added or removed during the training process. The subtleties are described in the following sections. In the software, the subroutine `gmix_trainscript.m` handles the details.

To illustrate the PDF estimation problem, we will use some 3-dimensional features from a mysterious source. Samples of the feature vector $\mathbf{z} = \{z_1, z_2, z_3\}$ were used as training data and were stored in variable `data1`, which is of size 3 by K, where K is the number of independent samples. Each row of the matrix stores the samples of a different feature. The following code segment implements the training and displays the resulting PDF in a density plot.

```
NMODE=10;
min_std  = [20  20  1.0];
names  = {'Z1','Z2','Z3'};

gparm1  = init_gmix(data1,NMODE,names,min_st          d);
for  i=1:100,
   [gparm1,Q]   = gmix_step(gparm1,data1);
   fprintf('%d:    Total  log-likelihood=%g\n',i,Q);
end;
gmix_view2(gparm1,data1,1,2);
```

Refer to table 4.3 for symbol names. The variable `names` is a cell array that stores the feature names for use in visualization plots. The variable `min_std` stores the minimum feature standard deviations (See section 4.3.4). The routine `init_gmix.m` creates an initial set of parameters. In simple problems, the mixture can be trained by repeated calls to `gmix_step.m` as shown. In more difficult problems, it is necessary to do more to insure that there are the right number of modes and that the algorithm is converging properly. A representative MATLAB program for training is `gmix_trainscript.m`, which in turn calls `gmix_step.m`, the subroutine that actually implements the E-M algorithm. We will discuss the use of `gmix_trainscript.m` in more detail in the following sections. Results of running the above code segment are shown in Figure 4.1. Visualization is accomplished by `gmix_view2.m` for any desired 2-dimensional plane. A routine `gmix_view1.m` is also available for projecting on one axis using a histogram. We will describe a complete example in more detail in section 4.3.7.

Repeat until convergence:

1. Compute data weights. For $i = 1, \ldots, L$:
$$w_{i,k} = \frac{\alpha_i \ \mathcal{N}(\mathbf{z}_k, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \ \gamma_k}{\sum_{i=1}^{L} \alpha_i \ \ \mathcal{N}(\mathbf{z}_k, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)}.$$
2. For $i = 1, \ldots, L$ let:
$$\alpha_i' = \sum_{k=1}^{N} w_{i,k}.$$
3. Update the means. For $i = 1, \ldots, L$:
$$\boldsymbol{\mu}_i = \frac{1}{\alpha_i'} \sum_{k=1}^{N} w_{i,k} \ \mathbf{z}_k.$$
4. Update the covariances. For $i = 1, \ldots, L$:
$$\boldsymbol{\Sigma}_i = \frac{1}{\alpha_i'} \sum_{k=1}^{N} w_{i,k} \ (\mathbf{z}_k - \boldsymbol{\mu}_i) \ (\mathbf{z}_k - \boldsymbol{\mu}_i)'.$$
5. Condition the covariances. There are two methods for doing this, the BIAS and CONSTRAINT methods. The following is the BIAS method: For $i = 1, \ldots, L$:
$$\{\boldsymbol{\Sigma}_i\}_{n,n} = \{\boldsymbol{\Sigma}_i\}_{n,n} + \rho_n^2, \quad n = 1, \ldots, P,$$
where ρ_n is the assumed measurement standard deviation for the n-th element of feature $\mathbf{z}$. The addition of this *a priori* information about the feature serves to prevent the covariance matrices from becoming singular. These constants ρ_n^2 must be chosen carefully. The topic will be discussed in detail in section 4.3.4. The CONSTRAINT method is described therein.
6. Update mode weights. For $i = 1, \ldots, L$:
$$\alpha_i = \frac{\alpha_i'}{\sum_{k=1}^{N} \gamma_k}.$$

Table 4.1: Update Equations for Gaussian Mixtures. This is representative only. Actual implementation requires attention to numerical issues discussed in the text.

```
|---------------------------------         ------  -----  -----  ------  -----  -|
| structure   gparm:                                                             |
|     ----------------------------------         -----  -----  ------  -----  -|
|     | structure   features(1...DIM):                                           |
|     |     ----------------------------------         -----  ------  -----  -|
|     |     | name:          character   string                                  |
|     |     |--------------------------------         -----  ------  -----  -|
|     |     | min_std:       real  number                                        |
|     |     |--------------------------------         -----  -----  ------  -----  -|
|     |     | structure   modes(1...NMODES):                                     |
|     |     |     ----------------------------------         -----  ------  -----  -|
|     |     |     | weight:            real  number                            |
|     |     |     |--------------------------------         -----  ------  -----  -|
|     |     |     | mean:              DIM-by-1   vector   of  real  numbers      |
|     |     |     |--------------------------------         -----  ------  -----  -|
|     |     |     | cholesky_covar:    DIM-by-DIM    vector   of  real  numbers   |
---------------------------------         ------  -----  -----  ------  -----  -
```

Table 4.2: GPARM structure definition

Parameter Name	Mathematical Symbol or Description $i \in [1 \dots L],\ \ n \in [1 \dots P]$
GM Parameters	
DIM =length(gparm.features)	P
NMODES =length(gparm.modes)	Number of GM components, L
gparm.modes(i).weight	α_i
gparm.modes(i).mean	$\boldsymbol{\mu}_i$
gparm.modes(i).cholesky _covar	$\mathbf{R}_i$
gparm.features(n).min _std	ρ_n
gparm.features(n).name	Feature Name
Other Variables	
N	Number of input samples, N
data	Training data, $\mathbf{z}$
data _wts	Data weights γ_k

Table 4.3: Table of correspondence between MATLAB variables and mathematical symbols used in the text.

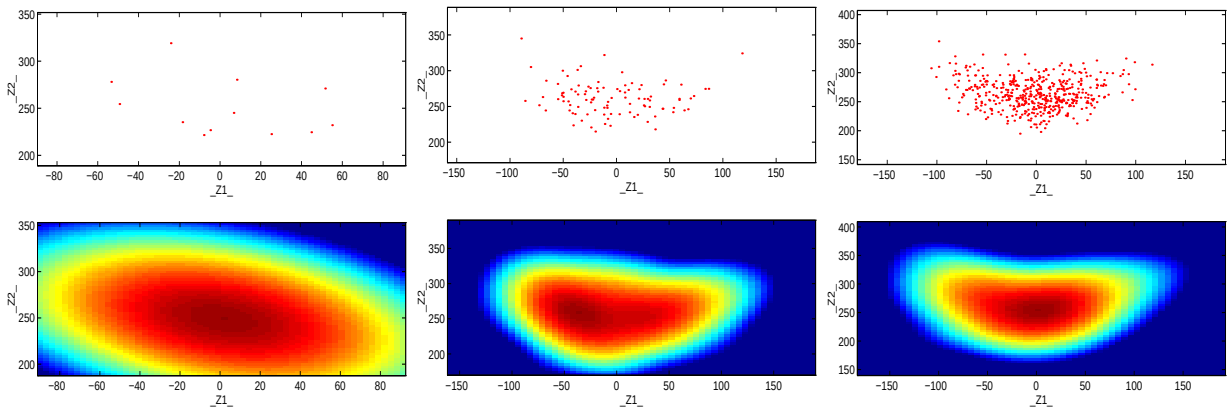

Figure 4.1: Results of PDF estimation for the 3-dimensional feature vector $\mathbf{z} = \{z_1, z_2, z_3\}$. Data and PDF's are projected on the (z_1, z_2) plane. The three cases are for 12, 100, and 500 training samples. The final number of mixture components (L) was 1, 6, and 8, respectively. The accuracy improves as the number of training samples increases.

Before iterating, a starting point is needed for the GM parameters. This is handled by `init_gmix.m` . This routine inputs some samples of data vectors $\mathbf{z}_1, \ldots, \mathbf{z}_N$, the number of GM terms to use (L), the covariance conditioning parameters ρ_n, and the names of all the features. The GM component means $\boldsymbol{\mu}_i$ are initialized to randomly selected input data samples. The covariances are initialized to diagonal matrices with large variances. It is important to use variances on the order of the square of the data volume width $|\max(z) - \min(z)|^2$. The size of the variances at initialization determines the data "window" through which each GM component "sees" the data. Too small a window at initialization can lock the algorithm into the wrong local minimum of the likelihood function. The initial weights α_i are set to be all equal.

There are two approaches to determining the number of modes. The first is to sprinkle a large number of modes throughout the data volume and remove the weak or redundant ones as it converges. The second approach is to start with just one mode and add modes as needed. The way you determine if a new mode is needed (by splitting an existing mode) is by a skew or kurtosis measure (`kurt.m`). These two methods, called *top-down* and *bottom-up*, respectively will be covered in section 4.3.5.

4.3.4 Implementation of the E-M algorithm : `gmix_step.m`

Working in the log domain.

Since probabilities can become extremely small, it is necessary to remain in the log-domain. Staying in the log-domain is a problem when summations are required. Let $l_i = \log \mathcal{N}(\mathbf{z}_k, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$. The summation

$$\log S = \log \left[\sum_{i=1}^{L} \alpha_i \exp(l_i) \right]$$

which appears in the first step of the E-M algorithm should be implemented as

$$\log S = M + \log \left\{ \sum_{i=1}^{L} \alpha_i \exp(l_i - M) \right\},$$

where $M = \max_i l_i$.

Using the Cholesky Decomposition of $\boldsymbol{\Sigma}_i$.

Instead of computing $\boldsymbol{\Sigma}_i$ directly, we store the Cholesky decomposition of $\boldsymbol{\Sigma}_i$ computed using the QR decomposition. Consider a matrix of column vectors $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]$. These columns correspond to the vectors $(\mathbf{z}_k - \boldsymbol{\mu}_i)$ in Table 4.1. A covariance estimate is obtained by forming the matrix $\boldsymbol{\Sigma} = \frac{1}{N}\mathbf{X}\mathbf{X}'$, which may be verified is the same as computing the elements of $\boldsymbol{\Sigma}$ as follows:

$$\boldsymbol{\Sigma}_{ij} = \frac{1}{N} \sum_{k=1}^{N} x_{ki}\, x_{kj}.$$

But note that if you take the QR decomposition $\mathbf{X}' = \mathbf{Q}\mathbf{R}$, that

$$\boldsymbol{\Sigma} = \frac{1}{N}\mathbf{X}\mathbf{X}' = \frac{1}{N}\mathbf{R}'\mathbf{Q}'\mathbf{Q}\mathbf{R} = \frac{1}{N}\mathbf{R}'\mathbf{R}.$$

Thus, we see that *the QR decomposition of $\mathbf{X}'$ is related to the Cholesky factor of $\boldsymbol{\Sigma}$*. There is no reason to ever compute $\boldsymbol{\Sigma}$ explicitly. Computing $\boldsymbol{\Sigma}$ requires twice the number of bits of precision as $\mathbf{R}$. A quadratic form can be computed using $\mathbf{R}$ as follows:

$$\mathbf{z}'\boldsymbol{\Sigma}^{-1}\mathbf{z} = \|\mathbf{y}\|^2$$

where

$$\mathbf{y} = \mathbf{z}'\mathbf{R}^{-1}.$$

This convention is used in the software (`gmix_step.m`). More precisely, the matrix `tmpidx` stores $\mathbf{X}'$ where the rows of $\mathbf{X}'$ are $(\mathbf{z}_k - \boldsymbol{\mu}_i)$. The QR decomposition of **tmpidx** is $\mathbf{R}$, which is stored as a parameter. The subroutine for computing $\log \mathcal{N}(\mathbf{z}_k, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$ is **lqr_eval.m**. This routine inputs $\mathbf{z}_1, \ldots, \mathbf{z}_N$, $\boldsymbol{\mu}_i$, and $\mathbf{R}_i$. The mixture (4.1) is implemented by subroutine **lqr_evp.m**.

Choosing the covariance constraints

If the quantization or additive measurement error variance is known for each feature, this can be used as a guide for choosing the covariance constraints. But, it can somewhat subjective if nothing is known about the data. A good idea of what to use for ρ_n may be obtained by observing the data on 2-dimensional projections. You should select ρ_n consistent with the width of the smallest visible cluster of data. For example, by looking at the top of Figure 4.3, ρ_1 and ρ_2 would be estimated by taking cross-sections of the visible data clusters along the X and Y axes, respectively. In the bottom of Figure, we see the result of choosing ρ_n too large (note the width of the small Gaussian mode is larger than the width of the corresponsing data cluster). It may be necessary to view the data in all possible 2-D projections before a decision can be made.

Conditioning the Covariances

Conditioning the covariances is accomplished without explicitly computing $\mathbf{\Sigma}_i$ as well. As mentioned in Table 4.1, step 5, there are two methods, the BIAS and CONSTRAINT methods. The BIAS method is simpler. On the other hand, the CONSTRAINT method delivers a better PDF estimate because the covariances are not biased and appears to converge faster. But, it may interfere with the monotonic increasing property of the E-M algorithm, i.e. that the total log-likelihood always goes up, but this is still an unresolved issue. Both methods are based on the idea of independent measurement error in the elements of $\mathbf{z}$. Let $\mathbf{D}$ be a diagonal covariance matrix with $\mathbf{D}_{n,n} = \rho_n^2$. The two methods differ in how they regard $\mathbf{D}$. The BIAS method assumes $\mathbf{D}$ is an *a priori* estimate of $\mathbf{\Sigma}$, while the CONSTRAINT method assumes $\mathbf{D}$ is a measurement error covariance.

The BIAS method is implemented by adding $\mathbf{D}$ to the newly formed covariance estimate. But, because we do not work with $\mathbf{\Sigma}$ directly, it is necessary to implement the conditioning as follows: Let $\mathbf{X}' = \mathbf{QR}$. The upper triangular matrix $\mathbf{R}$ is retained and $\mathbf{Q}$ is discarded. Next, we form the matrix as shown below for the case $P = 3$:

$$\mathbf{R}^* = \begin{bmatrix} \mathbf{R} \\ \cdots \\ \mathrm{diag}(\rho_n) \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ 0 & r_{22} & r_{23} \\ 0 & 0 & r_{33} \\ \cdots & & \\ \rho_1 & 0 & 0 \\ 0 & \rho_2 & 0 \\ 0 & 0 & \rho_3 \end{bmatrix}$$

It may be verified that $\mathbf{R}^{*\prime}\mathbf{R}^*$ is the same as $\mathbf{\Sigma}_i$ with the diagonal adjustments. Next, the QR-decomposition of $\mathbf{R}^*$ is computed and the upper triangular part is stored.

The CONSTRAINT method assumes that $\mathbf{\Sigma} = \mathbf{\Sigma}_0 + \mathbf{D}$ where $\mathbf{\Sigma}_0$ is an arbitrary covariance. Let the eigendecomposition of $\mathbf{\Sigma}$ be $\mathbf{\Sigma} = \mathbf{V}\mathbf{S}^2\mathbf{V}'$. Clearly, then

$$\mathbf{S}^2 = \mathbf{V}'\mathbf{\Sigma}_0\mathbf{V} + \mathbf{V}'\mathbf{D}\mathbf{V}.$$

Thus, the diagonal elements of $\mathbf{S}$ can be no smaller than the square root of the diagonal elements of $\mathbf{V}'\mathbf{D}\mathbf{V}$. Note that $\mathbf{V}$ and $\mathbf{S}$ may be obtained from the SVD of the Cholesky factor of $\mathbf{\Sigma}$:

$$\mathbf{\Sigma} = \mathbf{R}'\mathbf{R},$$

and

$$\mathbf{U}\mathbf{S}\mathbf{V}' = \mathbf{R}.$$

It is implemented in this way in `gmix_step.m` (`tmpvar` corresponds to $\mathbf{R}$):

```
[U,S,V]=svd(tmpvar,0);
 S = diag(S);
 S = max(S,sqrt(diag(     V' * diag(minvar)    * V )));
 tmpvar  = U * diag(S)  * V';
 [q,tmpvar]   = qr(tmpvar,0);
```

where the last two steps re-construct $\mathbf{R}$, then force it to be upper triangular.

Consider the following example. Data was created using a mixture of 2 Gaussians using the code segment below:

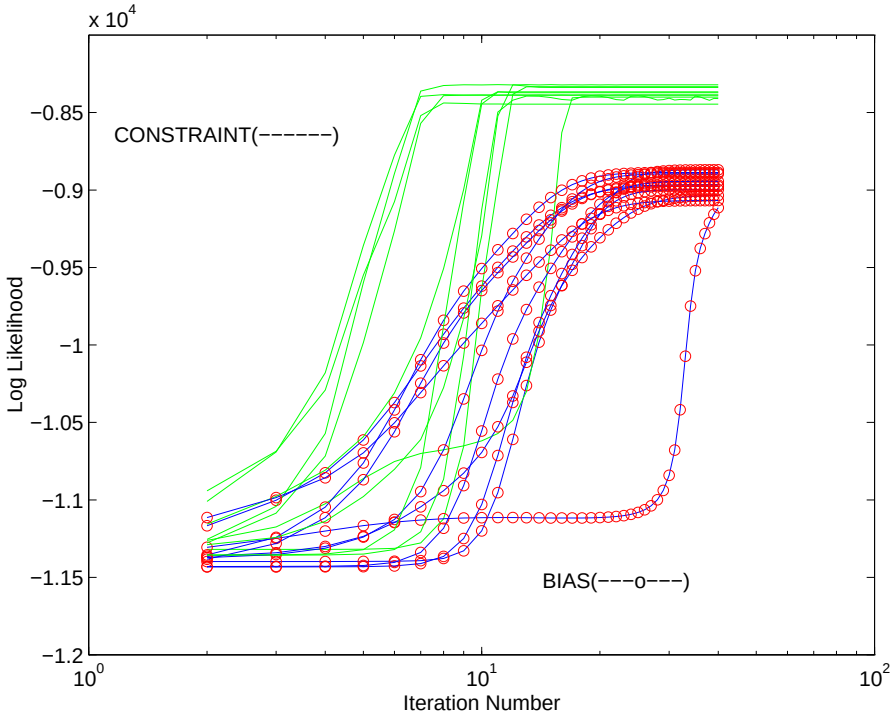

Figure 4.2: Convergence performance of the BIAS and CONSTRAINT methods. The CONSTRAINT method is consistently faster and achieves a higher log-likelihood every time.

```
%
% produce   data   that   is   from   two   Gaussian   populations
%
fprintf('Creating      data   : ');
N=4096;
mean1=[2    3]';
cov1=   [2  -1.6;   -1.6   2];
mean2=[1.3    1.3]';
cov2=   [.005   0;  0  .005];
x1  = chol(cov1)'     * randn(DIM,N);
x1=x1+repmat(mean1,1,N);
x2  = chol(cov2)'     * randn(DIM,N);
x2=x2+repmat(mean2,1,N);

data1   = [x1  x2];
```

Next, a GM parameter set was initialized with 2 modes with random starting means. Next, `gmix_step.m` was iterated 50 times using the BIAS and the CONSTRAINT method. This experiment was repeated 9 times. In each trial, the same starting point was used for both methods. The results are plotted in Figures 4.2 and 4.3. Note that the BIAS method has covariances that are biased and appear somewhat larger than necessary. In every case, the CONSTRAINT method converged faster and achieved a higher log-likelihood.

4.3.5 Training

Before training can occur, the GM paremeters must be initialized with a call to `init_gmix`, which was described in section 4.3.3, where we discussed two approaches to training. The *top-down* approach and *bottom-up* approaches are

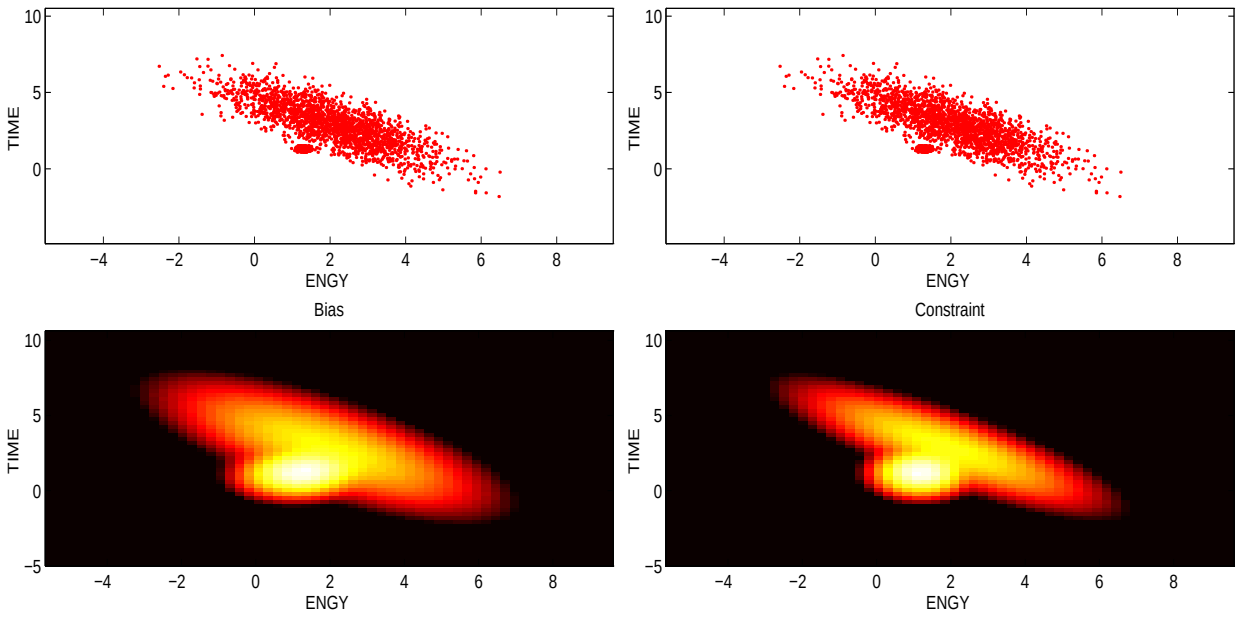

Figure 4.3: Typical results of training using the BIAS (left) and CONSTRAINT (right) methods. Each method used $\rho_n = 0.5$. Note that for the BIAS method, the covariance of the large mode is too fat, but for the CONSTRAINT method it is correct. For the small mode, the mode size is much smaller than ρ_n and therefore both methods produce similar results, as would be expected.

implemented simply by defining either a large number of modes or else just one mode, respectively. The number of modes is specified by in the arguments of `init_gmix.m` . But, training is more involved than just repeatedly calling `gmix_step.m` . Training involves five operations that are handled by `gmix_trainscript.m` :

1. E-M algorithm (`gmix_step.m`), sections 4.3.2,4.3.4.
2. Pruning modes (`gmix_deflate.m`), section 4.3.5.
3. Merging modes (`gmix_merge.m`), section 4.3.5.
4. Splitting modes (`gmix_kurt.m`), section 4.3.5.
5. Determining if algorithm has converged, section 4.3.5.

The operations are discussed in the indicated sections. An overall training script (`gmix_trainscript.m`) is discussed in section 4.3.5. The user has some control over some parameters used in training. In addition to the initial number of mixture modes, there are five other parameters that affect the training over which the user has some control.

1. The covariance constraints ρ_n (and selection of BIAS or CONSTRAINT method).
2. The minimum mode weight used in pruning modes.
3. The threshold used to determine if two modes should be merged.
4. The threshold to determine if a mode should be split.
5. The criterion for determining if convergence has occurred.

These parameters correspond directly to the five steps outlined above and are discussed in the indicated sections.

Determining the number of modes.

As we have stated, training can start with a large number of modes or just one mode. If the number of modes is too high, modes will be pruned out as α_i falls. If the number of modes is too low, modes will be split by `gmix_kurt.m`. Once the number of modes settles out and the likelihood stops increasing, convergence is declared.

The maximum number of modes to start with is about $N/(4P)$ where P is the dimension and N is the number of samples. If all the modes "share" the data equally, that is $4P$ samples per mode, a bare minimum. It is generally not problematic if the number of modes is over-specified since covariance estimates are stabilized by the conditioning discussed in section 4.3.4. And, as long as the amount of training data can support the number of modes chosen, the approximation is good. The mixing weight of a mode (α_i) multiplied by the number of input data samples N determines how many input data samples are effectively used to estimate the mode parameters. This is a simple measure of the "value" of each mode. As long as this product is high enough, the mode is estimated accurately. If α_i falls too low, the mode is eliminated or combined with another. With a combination of covariance constraints, pruning, merging, and mode splitting, a good PDF approximation can be obtained reliably.

E-M algorithm (`gmix_step.m`)

The E-M algorithm is explained in section 4.3.4. The calling syntax for `gmix_step.m` is as follows:

```
[gparm,Q]  = gmix_step(gparm,x,      [bias],[data_wts]);
```

where `gparm` are the input parameters, `x` is the normalized input data, `bias` (optional) is set to 1 for BIAS method and 0 for CONSTRAINT method, and `data_wts` (optional) allows individually weighting input data. On return, `Q` is the total log-likelihood.

Pruning (`gmix_deflate.m`)

Pruning is killing weak modes (a *mode* is another name for one of the L mixture components). A weak mode is found by testing α_i to see if it falls below a threshold. We have mentioned that $N\alpha_i$ is a measure of how many samples are "used" by mode i. To keep this quantity above kP, we require $\alpha_i > kP/N$. The quantity kP is called *SAMPLES_PER_MODE*, or `S_P_M` in the software. A good choice for k is about 4, so `S_P_M = 4*P`.

Pruning is handled by `gmix_deflate.m`. This program keeps bumping off the weakest mode and re-normalizing α_i so that $\sum_i \alpha_i = 1$. The calling syntax for `gmix_deflate.m` is

```
gparm  = gmix_deflate(gparm,min_weight_1,m        in_we ight_ all)
```

It is important that very weak modes be obliterated immediately, but it is important not to massacre lots of moderately weak modes all at once. So, there are two input thresholds. Only one mode per call to `gmix_deflate.m` can be bumped off if it falls below `min_weight_1`. But a mode is *always* bumped off if it falls below `min_weight_all`.

Merging Modes (`gmix_merge.m`)

Merging is creating a single mode from two nearly identical ones. The closeness of two modes is determined by `mode_dist.m` which works as follows. Let there be two PDF's $p_1(x)$ and $p_2(x)$. Let there be a collection of points denoted $x_i \in \mathbf{X}_1$ near the central peak of $p_1(x)$ and a collection of points denoted $x_i \in \mathbf{X}_2$ near the central peak of $p_2(x)$. Then we define the closeness metric

$$d = \log \left\{ \frac{\prod_{x_i \in \mathbf{X}_1} p_2(x_i) \prod_{x_i \in \mathbf{X}_2} p_1(x_i)}{\prod_{x_i \in \mathbf{X}_1} p_1(x_i) \prod_{x_i \in \mathbf{X}_2} p_2(x_i)} \right\}.$$

Notice that this metric is zero when $p_1(x) = p_2(x)$ and less that zero when $p_1(x) \neq p_2(x)$. A threshold (usually about -1 * DIM) is used to determine if the modes are too close. This threshold should increase (become more negative) as the dimension goes up.

Since $p_1(x)$ and $p_2(x)$ are just two Gaussian modes, it is easy to know where some good points for $\mathbf{X}_1$ and $\mathbf{X}_2$ are. We choose the means (centers) and then go one standard deviation in each direction along all the principal axes. The

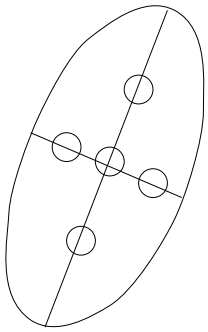

Figure 4.4: The 5 summation points for a 2-dimensional mode. Contour at 2σ.

principal axes are found by SVD decomposition of $\mathbf{R}$ (the Cholesky factor of the covariance matrix). This is illustrated in Figure 4.4 for a Gaussian mode of dimension $P = 2$. There is a center point and two points per dimension. Therefore there are $2P + 1$ points per mode, and two modes, thus $4P + 2$ points.

If two modes are found to be too close, they are merged. Merging is forming a weighted sum of two modes (weighted by α_1, α_2). The new mean is thus

$$\boldsymbol{\mu} = \frac{\alpha_1 \boldsymbol{\mu}_1 + \alpha_2 \boldsymbol{\mu}_2}{\alpha_1 + \alpha_2} \tag{4.3}$$

The proper way to form a weighted combination of the covariances is **not** simply a weighed sum of the covariances, which does not take into account the separation of the means. You need to be more clever. Consider the Cholesky decomposition of the covariance matrix $\boldsymbol{\Sigma} = \mathbf{R}'\mathbf{R}$. It is possible to consider the rows of $\sqrt{P}\ \mathbf{R}$ to be samples of P-dimensional vectors whose covariance is $\boldsymbol{\Sigma}$, where P is the dimension. The sample covariance is, of course $\frac{1}{P}(\sqrt{P})^2\ \mathbf{R}'\mathbf{R} = \boldsymbol{\Sigma}$, Now, given two modes to merge, we regard $\sqrt{P}\ \mathbf{R}_1$ and $\sqrt{P}\ \mathbf{R}_2$ as two populations to be joined. The sample covariance of the collection of rows is the desired covariance. But this assigns equal weight to the two populations. To weight them by their respective weights, we multiply them by $\sqrt{\frac{\alpha_1}{\alpha_1+\alpha_2}}$ and $\sqrt{\frac{\alpha_2}{\alpha_1+\alpha_2}}$. Before they can be joined, however, they must be shifted so they are re-referenced to the new central mean. Here is a summary of the method:

1. Let $\boldsymbol{\mu}$ be as in (4.3).
2. Let $\mathbf{R}_i$ be the Cholesky factor of $\boldsymbol{\Sigma}_i$, $i = 1, 2$.
3. Let $\mathbf{C}_i = \sqrt{P}\ \mathbf{R}_i$, each i.
4. Add the vector $\boldsymbol{\mu}_i - \boldsymbol{\mu}$ to each row of $\mathbf{C}_i$, each i.
5. Multiply $\mathbf{C}_i$ by $\sqrt{\frac{\alpha_i}{\alpha_1+\alpha_2}}$, each i.
6. Form
$$\mathbf{C} = \begin{bmatrix} \mathbf{C}_1 \\ \cdots \\ \mathbf{C}_2 \end{bmatrix}$$
7. Then the new covariance is $\boldsymbol{\Sigma} = \frac{1}{P}\ \mathbf{C}'\mathbf{C}$, or take the QR-decomposition of $\mathbf{C}/\sqrt{P}$ as the Cholesky factor of the new covariance.

The above algorithm is implemented by `merge.m` . The subroutine that iterates over all the pairs of modes and calls `merge.m` and `mode_dist.m` is `gmix_merge.m` . The calling syntax for `gmix_merge.m` is

```
gparm = gmix_merge(gparm,max_closeness)
```

A good choice for the `max_closeness` threshold is about -1.0 times P, the PDF dimension.

Splitting modes (`gmix_kurt.m`)

In a method proposed by N. Vlassis and A. Likas [38], the number of modes in a Gaussian mixture is determined by monitoring the weighted kurtosis for each mode. Putting their equation for one-dimensional $\mathbf{z}$ in our notation, Vlassis *et al* define

$$\kappa_i = \frac{\sum_{n=1}^{N} w_{n,i} \left(\frac{\mathbf{z}_n - \boldsymbol{\mu}_i}{\sqrt{\boldsymbol{\Sigma}_i}} \right)^4}{\sum_{n=1}^{N} w_{n,i}} - 3$$

where

$$w_{n,i} = \frac{\mathcal{N}(\mathbf{z}_n, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)}{\sum_{n=1}^{N} \mathcal{N}(\mathbf{z}_n, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)}$$

If $|\kappa_i|$ is too high for any mode i, they split the mode into two. We modify this for higher dimension and use the skew in addition to the kurtosis. Extending to higher dimension is done by projecting each data sample $\mathbf{z}_n$ onto the j-th principal axis of $\boldsymbol{\Sigma}_i$ in turn. Let $z_{n,i}^j \triangleq (\mathbf{z}_n - \boldsymbol{\mu}_i)' \mathbf{v}_{ij}$ where $\mathbf{v}_{ij}$ is the j-th column of $\mathbf{V}$, obtained from the SVD of $\boldsymbol{\Sigma}_i$ (see discussion in section 4.3.5). Thus, for each j,

1. Let
$$\kappa_{i,j} = \frac{\sum_{n=1}^{N} w_{n,i} \left(\frac{z_{n,i}^j}{s_i} \right)^4}{\sum_{n=1}^{N} w_{n,i}} - 3$$
2. Let
$$\psi_{i,j} = \frac{\sum_{n=1}^{N} w_{n,i} \left(\frac{z_{n,i}^j}{s_i} \right)^3}{\sum_{n=1}^{N} w_{n,i}}$$
3. Let
$$m_{i,j} = |\kappa_{i,j}| + |\psi_{i,j}|$$

where

$$s_i^2 = \frac{\sum_{n=1}^{N} w_{n,i} \left(z_{n,i}^j \right)^2}{\sum_{n=1}^{N} w_{n,i}}$$

Now, if $m_{i,j} > \tau$, for any j, split mode i. Split the mode by creating modes at

$$\boldsymbol{\mu} = \boldsymbol{\mu}_i + \mathbf{v}_{ij} S_{i,j}$$

and

$$\boldsymbol{\mu} = \boldsymbol{\mu}_i - \mathbf{v}_{ij} S_{i,j}$$

where $S_{i,j}$ is the j-th singular value of $\boldsymbol{\Sigma}_i$. The same covariance $\boldsymbol{\Sigma}_i$ is used for each new mode. Of course, the decision of whether to split or not depends on the mixing proportion α_i as well. No splitting occurs if α_i is too small.

In the following example, we create data with a gap in it. We begin iterating with a single mode. The kurtosis/skew algorithm above is able to assign modes until it is finally happy after 8 modes (Figure 4.5). The calling syntax for `gmix_kurt.m` is

```
gparm  = gmix_kurt(gparm,x,[kurt_thresh],[          debug ]);
```

The optional threshold parameter (default=1.0) allows control over splitting. A higher threshold is less likely to split. The optional debug parameter, if set to 1, will print out kurtosis and skew information.

Convergence

A good way to monitor the algorithm to detect convergence is to maintain a history list of the last few values of Q. If there is no improvement in Q for the duration of the history list, terminate the training. Note that because of pruning, etc, it is possible for Q to go down.

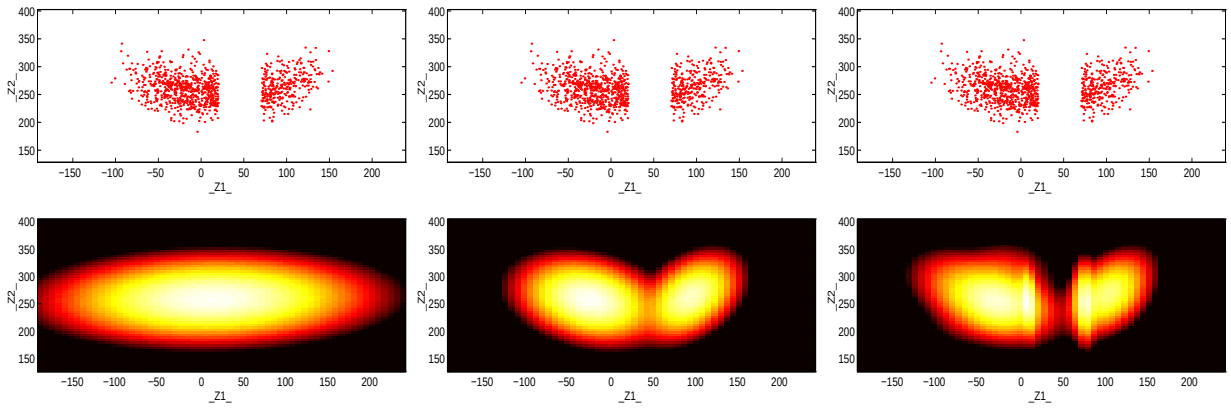

Figure 4.5: Results of *bottom up* PDF estimation. One mode (left), two modes (center), and after convergence at 8 modes (right).

Training script (`gmix_trainscript.m`)

The script `gmix_trainscript.m` may be used with the simple syntax:

```
gparm=gmix_trainscript(gparm,data,MAXIT);
```

where `gparm` is the GM parameter vector, `data` is the N-by-P input data vector, and `MAXIT` is the maximum number of iterations allowed. For added control, additional parameters may be added using the syntax

```
gparm=gmix_trainscript(gparm,data,MAXIT,SAMPLES_PER_MODE, BIAS, ...
                       max_close, addmodes, kurt_thresh)
```

The meaning of these parameters are discussed in previous sections.

Training on Huge data sets

If the number of data samples (N) is very large, the training scripts can choke like a chihuahua trying to eat a watermellon in one gulp. To handle this problem, there are scripts that can chop the watermellon into bite-size chunks and have the same effect as the whole watermellon. The relevant scripts are `gmix_accum.m` and `gmix_norm.m`. The following code demonstrates how to use these two routines in place of `gmix_step.m`.

```
%--------------------------------          ------ ----- -----
% Synopsis:  bite-size  replacement  for
%    [gparm,Q]  = gmix_step(gparm,xn);
%  The following  code  is equivalent  to one call to
%  gmix_step.    The numerical  behavior  is identical.
%--------------------------------          ------ ----- -----
%

gparm  = init_gmix(    .....);

for  iteration=1:10,

   % initialize   accumulators   to zero
   % at start  of each  iteration
   newmean=[];
   newvar=[];
```

```
    atot=zeros(nmode,1);
    for  i=1:nmode,
        newmean{i}=zeros(dim,1);
        newvar{i}=zeros(dim,dim);
    end;
    qtot=0;

    % Loop  over  1000  bite-size   pieces
    for  i=1  : 1000,
        x = ...       % get  new  data  matrix
        [newmean,newvar,atot,qtot]        = ...
            gmix_accum(gparm,x,newmean,newvar,ato          t,qto t);
    end;

    % finalize   the  iteration
    gparm  = gmix_norm(gparm,newmean,newvar,atot          );
end;
```

4.3.6 Conditional PDFs and Conditional Mean using Gaussian Mixtures

Gaussian mixtures afford a convenient way to generate conditional PDFs and conditional mean estimates.

Conditional Estimation in general

Let the data vector $\mathbf{z}$ be composed of two parts $\mathbf{x}$ and $\mathbf{y}$:

$$\mathbf{z} = \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix}.$$

We have available training samples of $\mathbf{z}$, however in the future, only $\mathbf{y}$ will be available from which we would like to compute estimates of $\mathbf{x}$. We will shortly see that the GM density facilitates the computation of the *conditional mean* or *minimum mean square error* (MMSE) estimator of $\mathbf{x}$. The conditional mean estimator is the expected value of $\mathbf{x}$ conditioned on $\mathbf{y}$ taking a specific (measured) value, i.e.,

$$\hat{\mathbf{x}} = \mathbf{E}(\mathbf{x}|\mathbf{y}) = \int_{\mathbf{x}} \mathbf{x}\, p(\mathbf{x}|\mathbf{y})\, d\mathbf{x}$$

The maximum aposteriori (MAP) estimator is given by

$$\hat{\mathbf{x}} = \max_{\mathbf{x}} p(\mathbf{x}|\mathbf{y}).$$

Both the MAP and MMSE estimators are operations performed on $p(\mathbf{x}|\mathbf{y})$. Which estimator is most appropriate depends on the problem. Suffice it to say that the distribution $p(\mathbf{x}|\mathbf{y})$ expresses all the knowledge we have about $\mathbf{x}$ after having measured $\mathbf{y}$.

Estimation using Gaussian Mixtures

The GM representation of the density has the a remarkable property that $p(\mathbf{x}|\mathbf{y})$ can be computed in closed form. This is especially useful in visualization of information. For example, it is useful to show a human operator the *distribution* of likely $\mathbf{x}$ after $\mathbf{y}$ is measured. If desired, the MMSE can be computed in closed form as well. The MAP estimate can also be computed, but that requires a search over $\mathbf{x}$.

Let the GM approximation to the distribution be given by

$$p(\mathbf{x}, \mathbf{y}) = \sum\nolimits_i \alpha_i p_i(\mathbf{x}, \mathbf{y}). \tag{4.4}$$

By Bayes rule,

$$p(\mathbf{x}|\mathbf{y}) = \frac{p(\mathbf{x}, \mathbf{y})}{p(\mathbf{y})} = \frac{1}{p(\mathbf{y})} \sum\nolimits_i \alpha_i p_i(\mathbf{x}, \mathbf{y})$$

where $p(\mathbf{y})$ is the *marginal* distribution of $\mathbf{y}$. We now define $p_i(\mathbf{y})$ as the marginal distributions of $\mathbf{y}$ given that $\mathbf{y}$ is a member of mode i. These are, of course, Gaussian with means and covariances taken from the $\mathbf{y}$-partitions of the mode i mean and covariance $\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i$.

$$\boldsymbol{\mu}_i = \begin{bmatrix} \boldsymbol{\mu}_{x,i} \\ \boldsymbol{\mu}_{y,i} \end{bmatrix} \quad \boldsymbol{\Sigma}_i = \begin{bmatrix} \boldsymbol{\Sigma}_{xx,i} & \boldsymbol{\Sigma}_{xy,i} \\ \boldsymbol{\Sigma}_{yx,i} & \boldsymbol{\Sigma}_{yy,i} \end{bmatrix}$$

Then,

$$\begin{aligned} p(\mathbf{x}|\mathbf{y}) &= \frac{1}{p(\mathbf{y})} \sum\nolimits_i \alpha_i p_i(\mathbf{y}) \frac{p_i(\mathbf{x},\mathbf{y})}{p_i(\mathbf{y})} \\ &= \frac{1}{p(\mathbf{y})} \sum\nolimits_i \alpha_i p_i(\mathbf{y}) p_i(\mathbf{x}|\mathbf{y}) \end{aligned} \tag{4.5}$$

where $p_i(\mathbf{x}|\mathbf{y})$ is the conditional density for $\mathbf{x}$ given $\mathbf{y}$ assuming that $\mathbf{x}$ and $\mathbf{y}$ are from that certain Gaussian sub-class i. Fortunately, there is a closed-form equation for $p_i(\mathbf{x}|\mathbf{y})$ [39]. $p_i(\mathbf{x}|\mathbf{y})$ is Gaussian with mean

$$\mathbf{E}_i(\mathbf{x}|\mathbf{y}) = \boldsymbol{\mu}_{x,i} + \boldsymbol{\Sigma}_{xy,i} \boldsymbol{\Sigma}_{yy,i}^{-1} (\mathbf{y} - \boldsymbol{\mu}_{y,i}). \tag{4.6}$$

and covariance

$$\mathrm{cov}_i(\mathbf{x}|\mathbf{y}) = \boldsymbol{\Sigma}_{xx,i} - \boldsymbol{\Sigma}_{xy,i} \boldsymbol{\Sigma}_{yy,i}^{-1} \boldsymbol{\Sigma}_{yx,i}. \tag{4.7}$$

Note that the conditional distribution is a Gaussian Mixture in its own right, with mode weights modified by $p_i(\mathbf{y})$ which tends to "select" the modes closest to $\mathbf{y}$. To reduce the number of modes in the conditioning process, one could easily remove those modes with a low value of $p_i(\mathbf{y})$ (suggested by R. L. Streit).

This conditional distribution can be used for data visualization or, to easily calculate the conditional mean estimate, which is a by-product of equations (4.5),(4.6),(4.7):

$$\begin{aligned} \mathbf{E}(\mathbf{x}|\mathbf{y}) &= \int_{\mathbf{x}} p(\mathbf{x}|\mathbf{y})\ \mathbf{x}\ d\mathbf{x} \\ &= \frac{1}{p(\mathbf{y})} \sum\nolimits_i \alpha_i p_i(\mathbf{y}) \int_{\mathbf{x}} p_i(\mathbf{x}|\mathbf{y})\ \mathbf{x}\ d\mathbf{x} \\ &= \frac{1}{p(\mathbf{y})} \sum\nolimits_i \alpha_i p_i(\mathbf{y}) \mathbf{E}_i(\mathbf{x}|\mathbf{y}) \end{aligned} \tag{4.8}$$

MATLAB implementation

The subroutine `gmix_condx.m` is used to generate the conditional distribution. The syntax is

```
gparm2 = gmix_condx(gparm,x_idx,y_idx,y)
```

where `gparm` is the GM parameter vector for $p(\mathbf{z})$, `x_idx` are the indexes indicating which elements of $\mathbf{z}$ constitute $\mathbf{x}$ (they can be any elements), and similarly for `y_idx` .

The subroutine `gmix_cmean.m` uses `gmix_condx.m` to compute the conditional mean of $\mathbf{x}$. The syntax is

```
xhat = gmix_cmean(gparm,x_idx,y_idx,y)
```

where all inputs are identical to `gmix_condx.m` . The one exception is that input `y` can include any number of samples of $\mathbf{y}$. The dimensions of `y` are N-by-P where N is the number of samples and P is the dimension of $\mathbf{y}$.

Example of Estimation: Beam Interpolation

Assume that beam intensity values are available from a set of M uniformly spaced (in direction) sonar or radar beams. A target exists somewhere in the span of the M beams, yet we do not know its center location, nor do we know the width of the response to the signal (as in a broadband system with frequency-dependent beamwidth). We assume for simplicity that the amplitude is known, yet in principle, amplitude can be another unknown. Thus, there are two parameters we seek to estimate: direction d and beamwidth w. This problem normally requires a search in the d, w plane for best match (as in maximum likelihood). Using GM, we solve the problem without a search, yet achieve performance comparable to ML!

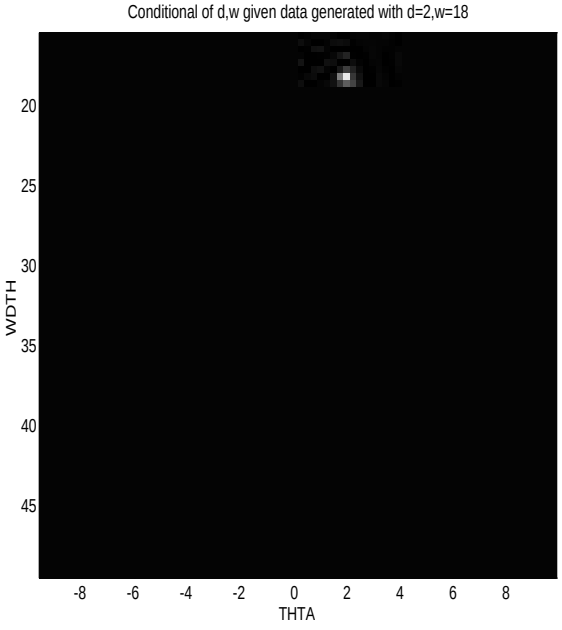

Figure 4.6: Condition distibution of d (THTA) and w (WDTH) given a sample of $\mathbf{b}$ computed for $d = 2, w = 18$ with no additive noise.

Let the beam pointing directions be $d_1, \ldots, d_M$. Let the beam intensities $\mathbf{b} = \{b_1 \ldots b_M\}$ be modeled by

$$b_i = A \exp\left\{-0.346(d - d_i)^2 \frac{4}{w^2}\right\} + n_i$$

where n_i is a noise term (we use Gaussian noise in the simulation and CR bound analysis). This is a Gaussian beampattern with 3 dB width w.

A sample size of 4096 was created using d and w selected from uniform distributions in the ranges [-10,10], [15,50], respectively. Parameters were $A = 50$, $\sigma^2 = 1$, $M = 5$, $\{\theta_i\} = \{-20, -10, 0, 10, 20\}$. A GM model $p(\mathbf{b}, d, w)$ of 12 modes was trained on the data. To illustrate the ability to create conditional distributions, $p(d, w|\mathbf{b})$ was computed for a sample of $\mathbf{b}$ computed for $d = 2, w = 18$ with no additive noise. The result appears in Figure 4.6. The visual effect of this figure is to say to the operator that there are no other values of interest except the peak.

It is also possible to condition on d or w. The conditional distribution $p(\mathbf{b}, w|d)$ was computed for $d = 0$ and $d = -5$. these plots are shown in Figures 4.7,4.8. Note that the beam output values have distributions symmetric about the value of d, as expected. Note also the wider spread of values on outer beams due to the variations in w.

Estimates of d, w were obtained using formulas (4.8),(4.6). To determine bias, uncorrupted (no noise) values of $\mathbf{b}$ were created for a range of d for w fixed at 20, and for a range of w for d fixed at 2. These two graphs appear in Figures 4.9,4.10. In each case, the bias error is plotted as a function of the variable parameter. Bias is clearly a function of the operating point. It is also a function of the number of modes and the convergence point of the GM approximation algorithm. Random error was determined by choosing a specific value of d, w and running 300 trials with independent noise added to $\mathbf{b}$. The result of 300 trials is shown below.

	True Value	Mean	Variance	CR Bound
d	2	1.9435	.0550	.0493
w	18	18.003	.09756	.0945

Results of 300 trials, $A = 50,\ \ \sigma^2 = 1,\ M = 5.$

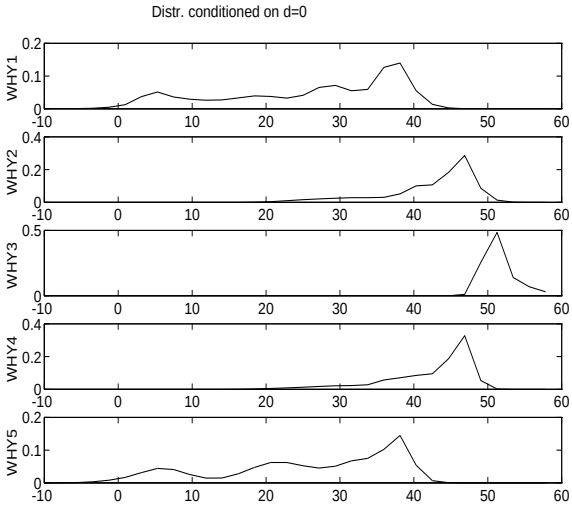

Figure 4.7: The condition distibution $p(\mathbf{b}, w|d)$ marginalized on each dimension of $\mathbf{b}$ for $d = 0$.

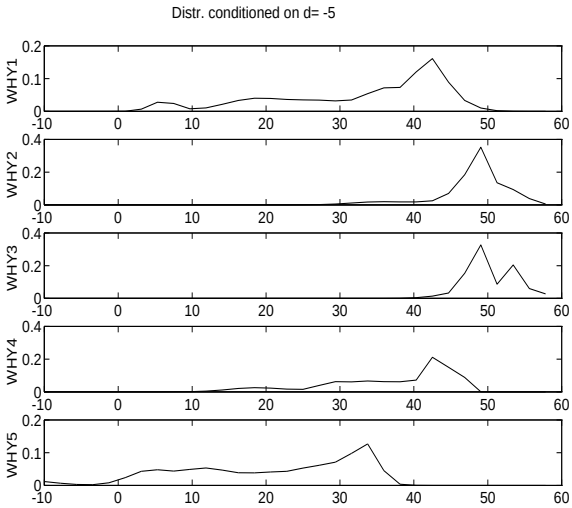

Figure 4.8: The condition distibution $p(\mathbf{b}, w|d)$ marginalized on each dimension of $\mathbf{b}$ for $d = -5$.

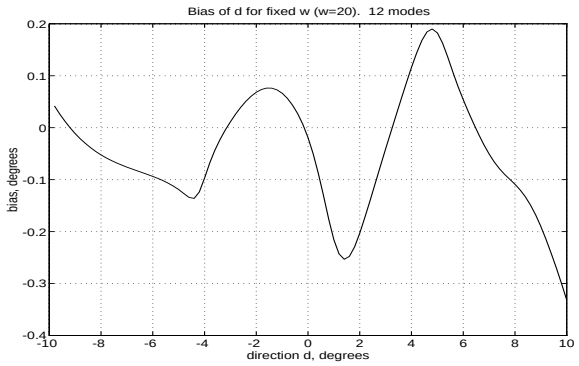

Figure 4.9: Plot of $\hat{d} - d$ for noise-free data with $w = 20$.

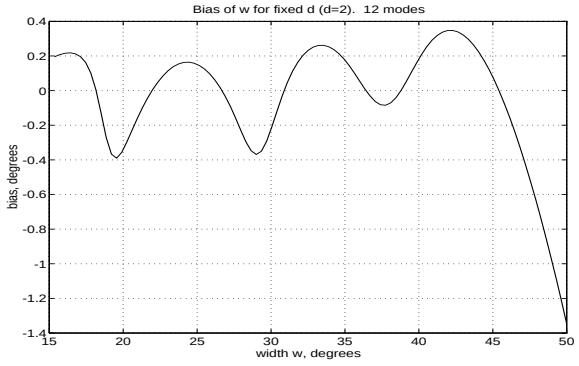

Figure 4.10: Plot of $\hat{w} - w$ for noise-free data with $d = 2$.

The results were in close agreement with the CR bound. Strictly speaking, the CR bound does not apply since the conditional mean estimator is biased for a fixed d, w (it is unbiased for random d, w conditioned on $\mathbf{b}$), however, the CR bound is useful for comparison purposes.

CR Bound analysis

The log-PDF of the data $\mathbf{b}$ is

$$\ln p(\mathbf{b}; d, w) = -\frac{1}{2}\ln(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{M}\left[b_i - A\exp\left\{-0.346(d-d_i)^2\frac{4}{w^2}\right\}\right]^2,$$

where σ^2 is the variance of the additive independent Gaussian noise. The components of the Fisher Information Matrix (FIM) for PDF parameters ϕ_i, ϕ_j are given by

$$F_{\phi_i,\phi_j} = -\mathbf{E}\left(\frac{\partial^2 \ln p(\mathbf{b}; \phi_i, \phi_j)}{\partial\phi_i \partial\phi_j}\right)$$

Let the FIM be given by

$$F(d, w) = \begin{bmatrix} F_{dd} & F_{dw} \\ F_{wd} & F_{ww} \end{bmatrix}.$$

A standard CR bound analysis [40] produces

$$F_{dd} = \frac{A^2}{\sigma^2}\left(0.346\frac{8}{w^2}\right)^2 \sum_{i=1}^{M}\left((d-d_i)\exp(-\omega_i)\right)^2$$

$$F_{ww} = \frac{A^2}{\sigma^2}\left(0.346\frac{8}{w^3}\right)^2 \sum_{i=1}^{M}\left((d-d_i)^2\exp(-\omega_i)\right)^2$$

$$F_{dw} = F_{wd} = \frac{A^2}{w\sigma^2}\left(0.346\frac{8}{w^2}\right)^2 \sum_{i=1}^{M}\left((d-d_i)^2\exp(-\omega_i)\right)^2$$

where $\omega_i = 0.346(d-d_i)^2\frac{4}{w^2}$. The CR bound matrix is given by $\mathbf{C}(d, w) = \mathbf{F}^{-1}(d, w)$.

4.3.7 An Example Script for Gaussian Mixtures

Script `gmix_example.m` is designed as a teaching example for use of the software. All the basic functions as well as some handy utilities are demonstrated. Refer to the program listing for the discussion that follows. After typing `>> gmix_example` at the MATLAB prompt, you will see the graph of Figure 4.11 and the program will pause. This is a two-dimensional "point scatter" diagram of the data that we will fit a Gaussian Mixture to. Refer to the program listing to see how this data is created. Pressing any key initializes the Gaussian Mixture parameters with the following 3 lines:

```
names={'ENGY','TIME'};
min_std  = [.1  .1];
NMODE=1;
gparm1=init_gmix(data1,NMODE,names,min_std) ;
```

The first line assigns names to the two dimensions. We have chosen to call them "ENGY" and "TIME". The next line assigns the ρ_n parameters, as discussed in section 4.3.4. The N training data samples are stored in the $P \times N$ variable `data1` . The last line creates the parameter structure `gparm1` . Since the algorithm starts with just a single mode (`NMODE=1`), the approximation is poor (Figure 4.12). Pressing a key again executes the training with a 150-iteration limit:

```
gparm1=gmix_trainscript(gparm1,data1,150) ;
```

The log likelihood (`Q`) is printed out at each iteration along with the number of modes. Log likelihood would monotonically increase, if not for the pruning, splitting, and merging operations. Use of the CONSTRAINT method of covariance conditioning will also affect the monotonicity. It may be verified, however, that calls to `gmix_step.m` with BIAS=1

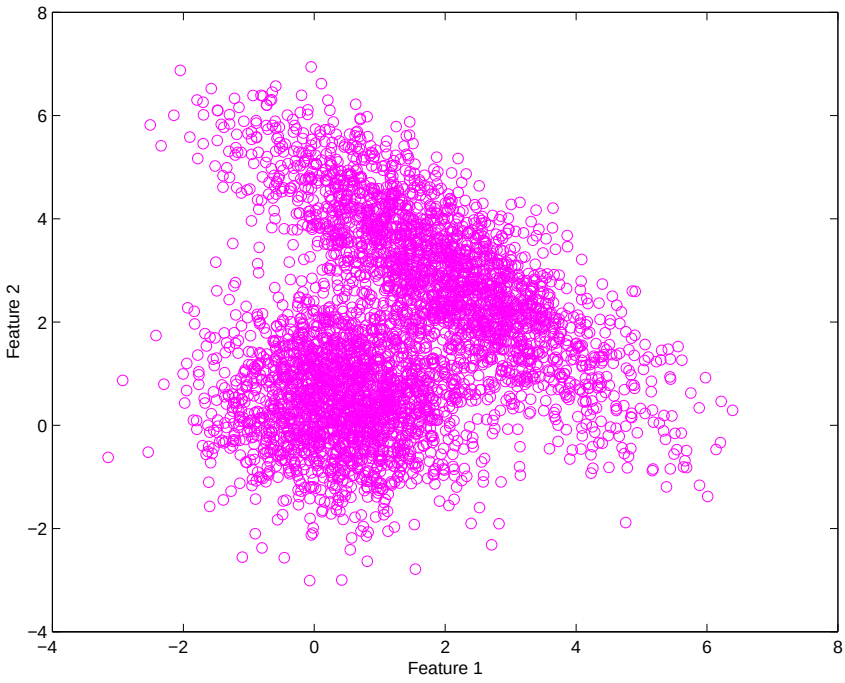

Figure 4.11: Samples from a Gaussian mixture.

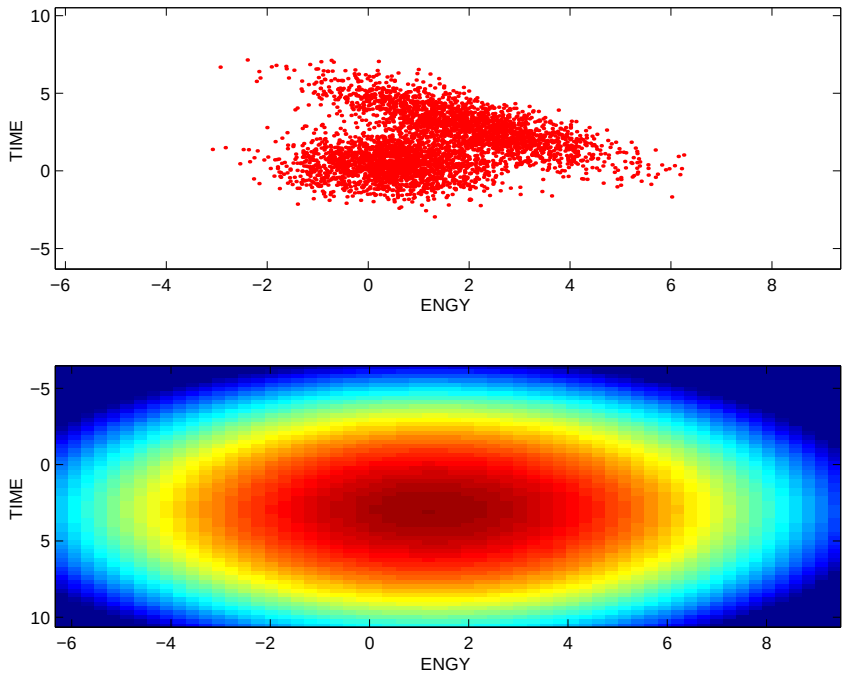

Figure 4.12: The initial Gaussian mixture approximation.

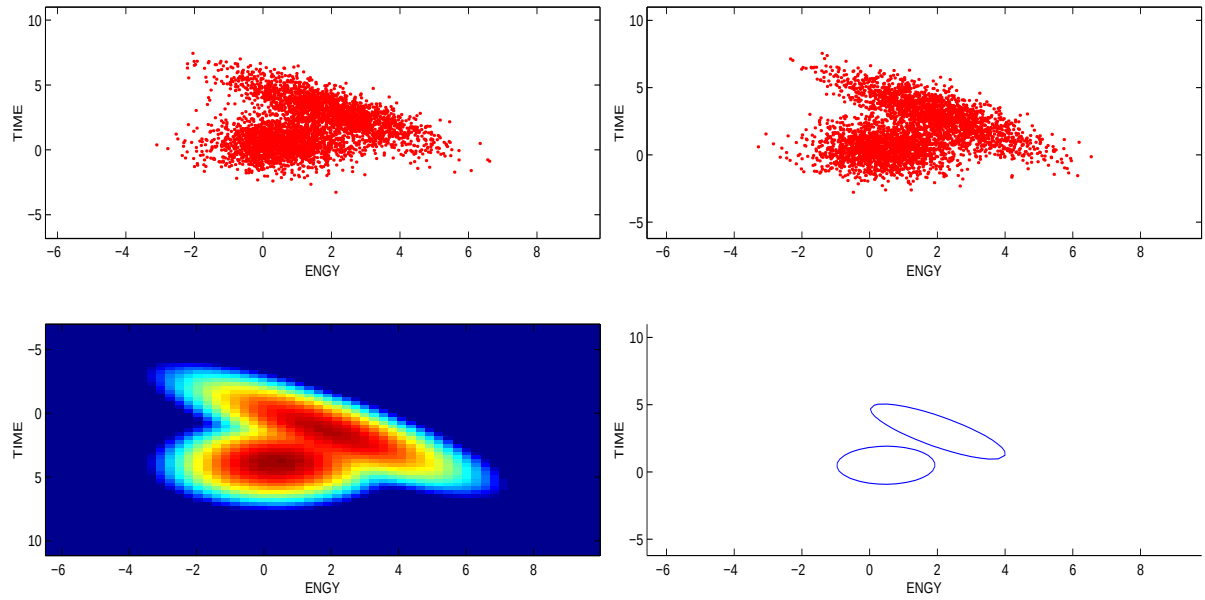

Figure 4.13: Gaussian mixture approximation after convergence.

will result in monotonic likelihood increase, with the possible exception of numerical errors at the very end of the convergence process. Whenever mode splitting occurs, the message "Adding a mode .." is printed. Whenever mode merging occurs, the message "Merging ..." is printed. Because in this example, we have initialized with just one mode, mode splitting is more likely that merging, although is is possible that after several modes have been split, they can be remerged. This causes a "fight" between `gmix_kurt.m` which tries to split, and `gmix_merge.m`, which tries to merge. In `gmix_trainscript.m`, it is arranged to allow time between splitting and merging so that the E-M algorithm can settle out. Otherwise, newly merged modes could be quickly split, or newly split modes could be quickly merged.

Once `gmix_trainscript.m` converges, you should should see the graph on the left of Figure 4.13. These plots are produced by `gmix_view2.m`. This utility is perhaps the most useful visualization tool for high-dimensional PDF estimation. It allows the data scatter plot to be compared with the marginalized PDF on any 2-dimensional plane.

Marginalization is a simple matter for Gaussian mixtures. Let $\mathbf{z} = [z_1, z_2, z_3, z_4]$. To visualize on the (z_2, z_4) plane, for example, we would need to compute

$$p(z_2, z_4) = \int_{z_1} \int_{z_3} p(z_1, z_2, z_3, z_4) dz_1 dz_3.$$

Instead of integrating out z_1, z_3, marginalization requires only stripping out the first and third elements of each mode mean vector, and the first and third rows and columns of each mode covariance, then computing the resulting Gaussian mixture! Because we work with the Cholesky decomposition of the mode covariances, it requires stripping out the necessary columns, then doing a QR decomposition of the result. This stripping operation is performed by `gmix_strip.m`. The syntax would be:

```
gparm_out = gmix_strip(gparm_in, [2 4]);
```

where the second argument indicates that we want to retain the second and fourth dimensions. Using this method, the marginal distribution of any 2-dimensional plane is easily computed. Stripping is handled automatically by `gmix_view2.m`.

Press once more and the intensity plot is replaced by a contour plot of the modes (on the right of Figure 4.13. The contour plot is obtained by the fourth argument to `gmix_view2.m`. The complete syntax of `gmix_view2.m` is

```
[p,xp,yp]=gmix_view2(gparm1,data1,idx1,idx2,[do_ellip],[M],[plot]);
```

where `idx1, idx2` are the indexes of the dimensions requested and `do_ellip` is an optional argument that, if equal to 1 (default=0), produces a contour plot of each mode instead of an image. `M` is an optional parameter that defines

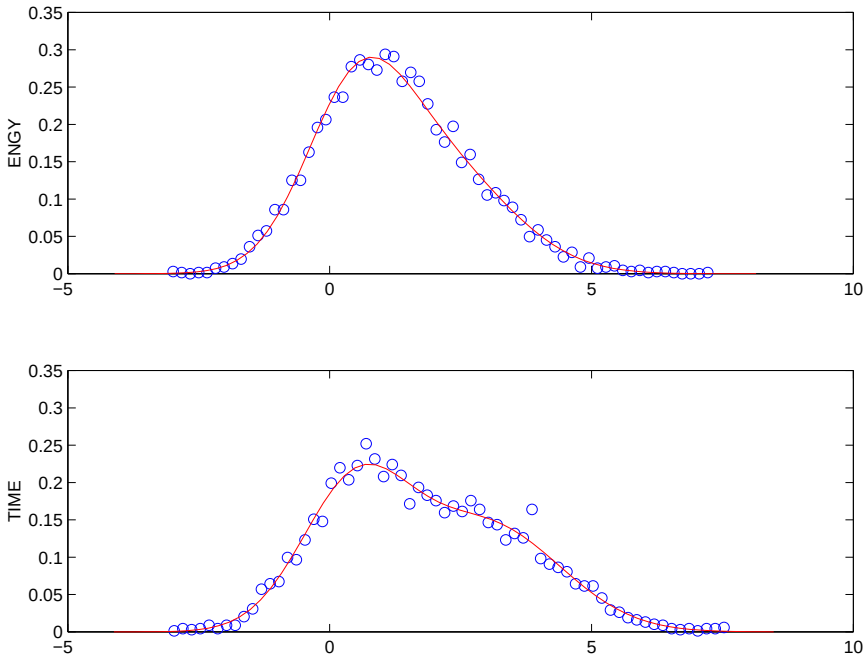

Figure 4.14: One dimensional PDF plots; Marginal PDF's compared to histograms.

the number of resolution cells for each dimension of the plot (default=60). The optional parameter `iplot` can be set to zero if only the outputs `p,xp,yp` are wanted. These outputs provide the output PDF grid that can be plotted by `imagesc(xp,yp,p)`. In the example, since the data is 2-dimensional to begin with, there is no dimension reduction performed by `gmix_view2.m`.

Information about the Gaussian mixture may be printed by calling `gmix_show.m`. This information, which includes the mode weights, means, and determinants, can be directly compared with Figure 4.13. The true means are (2,3) and (.5,.5), and the true determinants are 1.44 and 1.0, respectively. Generally, if the algorithm results in just 2 modes, the parameters agree very closely. The inclusion of a third or fourth mode makes it difficult to see the correspondence. But, nevertheless, the PDF approximation is good as evidenced by the intensity plot. You can run `gmix_example.m` again and each time the result will be a little different. But always, the intensity plot and the PDF approximation is excellent.

Press the key once more and Figure 4.14 will be plotted. This figure shows the 1-dimensional marginals for each dimension displayed along with the histograms. It is the result of calling `gmix_view1.m`. The calling syntax is

```
[pdf,xp,h,xh]=gmix_view1(gparm,data,idx,nbins);
```

If called without any output arguments, the plot will automatically be generated. Input `idx` is an array of indexes for the dimensions desired. For more than one index, multiple plots are produced. Input "nbins" is the histogram size.

Press a key once more and Figure 4.15. This figure demonstrates the function `gmix_cond` (See Section 4.3.6) which creates conditional PDFs from the original Gaussian mixture. Rather than using Bayes rule explicitly to compute the conditional PDF of x given that $y = y_0$,

$$p(x|y = y_0) = \frac{p_{xy}(x, y_0)}{p_y(y_0)},$$

it solves for the Gaussian mixture parameters of $p(x|y = y_0)$ in closed form so that you can later evaluate $p(x|y = y_0)$ at any point x, or else examine the parameters of the mixture. The figure shows the PDF evaluated at four values of y_0.

Press a key once more and Figure 4.16 is shown. This figure plots the original data again in green and some synthetic data created by `gmix_makedata.m` in red. This demonstrates a convenient aspect of GM approximation: generating synthetic data is simple.

Press a key once more and Figure 4.17 appears. We now have two data sets. We will now build a classifier using Gaussian mixtures. The first step is to train a second parameter set on the second data set. This time, we will use the

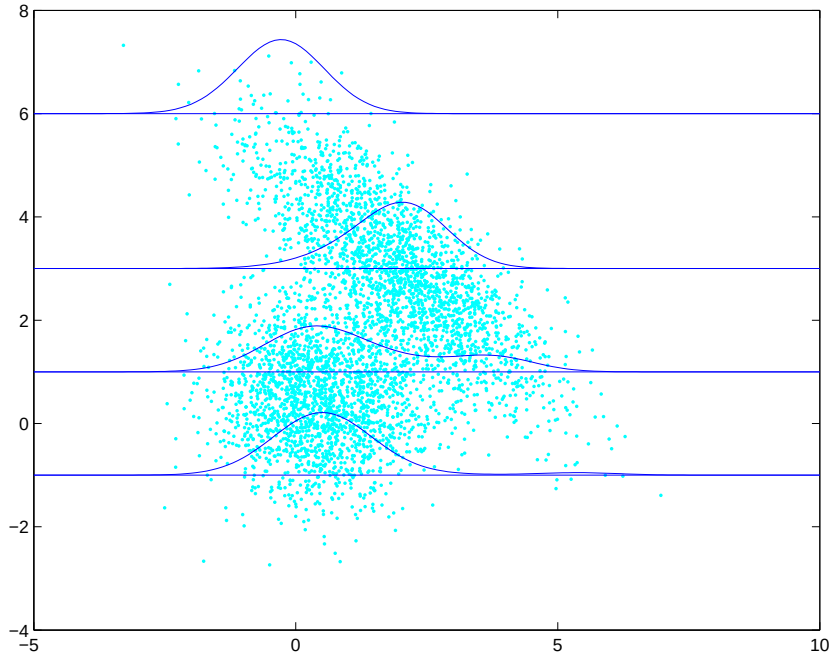

Figure 4.15: Conditional PDFs of x given $y = y_0$, at various values of y_0: {-1, 1, 3, 6} .

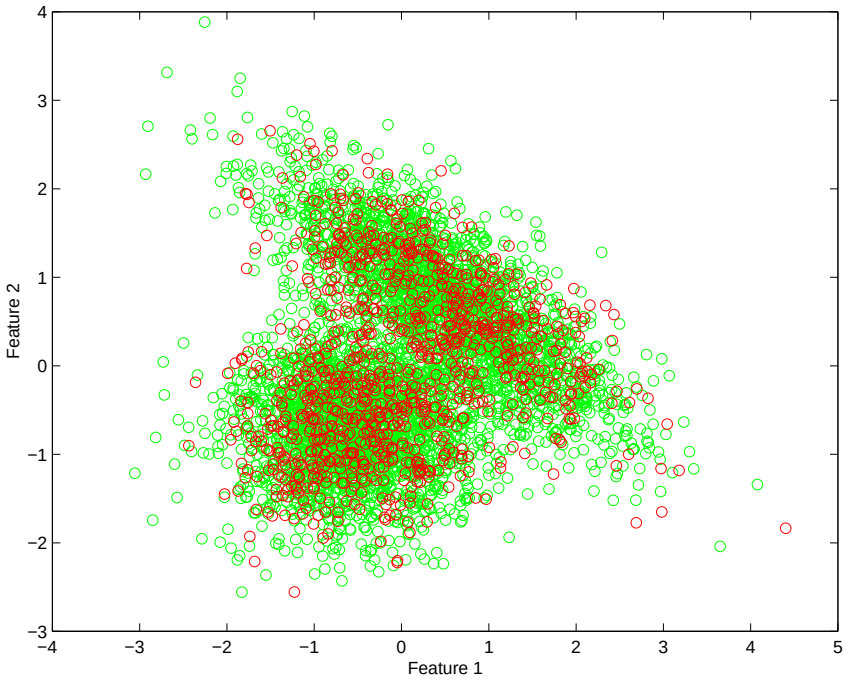

Figure 4.16: Original data (green) and synthetic data (red).

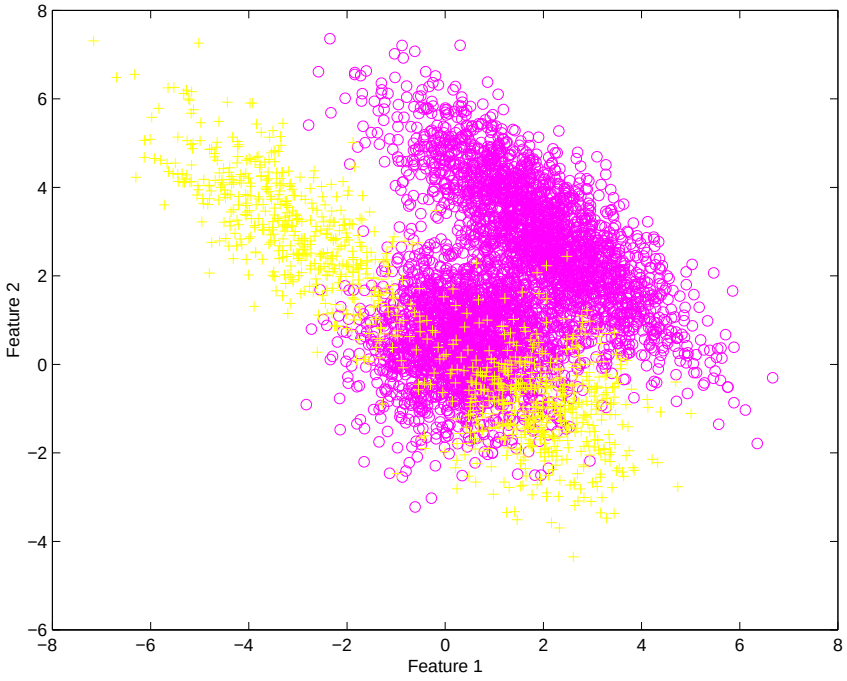

Figure 4.17: A second data class in yellow. The first data set in magenta.

top-down approach by initializing with 15 mixture modes. This time, there will be alot of merging and purging going on, but less splitting! Press a key again and the training starts. When complete, you should see Figure 4.18.

To classify, it is necessary to compute the log-likelihood of test data. This is done using `lqr_evp.m` . The name of the subroutine was not thought up logically, but evolved from (l)og-likelihood (ev)aluation from the (p)arameters, and the fact that the (QR) decomposition is involved in the covariance estimates! The calling syntax is

```
loglik  = lqr_evp(gparm1,data1,0);
```

If the third argument was 1, the routine would return a matrix of log-likelihoods where each column is from one of the mixture modes. The zero forces the modes to be combined with the apropriate weights into the GM approximation. The ROC curve is shown in Figure 4.19. Refer to the listing for details.

4.4 PDF Estimation using HMMs

The hidden Markov model (HMM) is a powerful statistical model that closely approximates many phenomenon found in nature, such as human speech. While a very powerful statistical model, a single HMM cannot easily act as a classifier between a wide variety of signal classes. Instead, it is best to design them specifically for each signal type and feature type. A versatile HMM software toolbox for MATLAB is also described.

4.4.1 Introduction to HMM's

The fundamental assumption of an HMM is that the process to be modeled is governed by a finite number of states and that these states change once per time step in a random but statistically predictable way. To be more precise, let $\Pr(q_t = i)$ be the probability that the system transitions into state i at time t. The Markovian assumption says that $\Pr(q_t = i)$ depends only on q_{t-1}, the true state at time $t-1$. Furthermore, if this distribution does not depend on the absolute time t, then the state probabilities can be described completely by a fixed *state transition matrix* $\mathbf{A} = \{a_{ij}\}$ where $a_{ij} = \Pr(q_t = j | q_{t-1} = i)$. Figure 4.20 illustrates a hidden Markov model (HMM). At each time step (time running from left to right), the Markov model is in one of the five possible states. According to the Markovian assumption, the

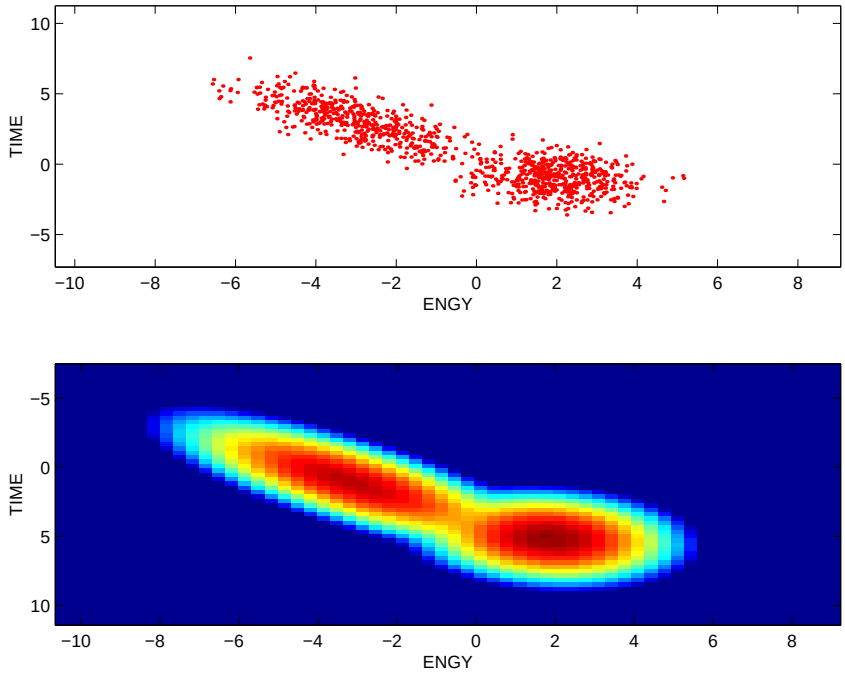

Figure 4.18: Result of top-down approach: Trained GM approximation of second class.

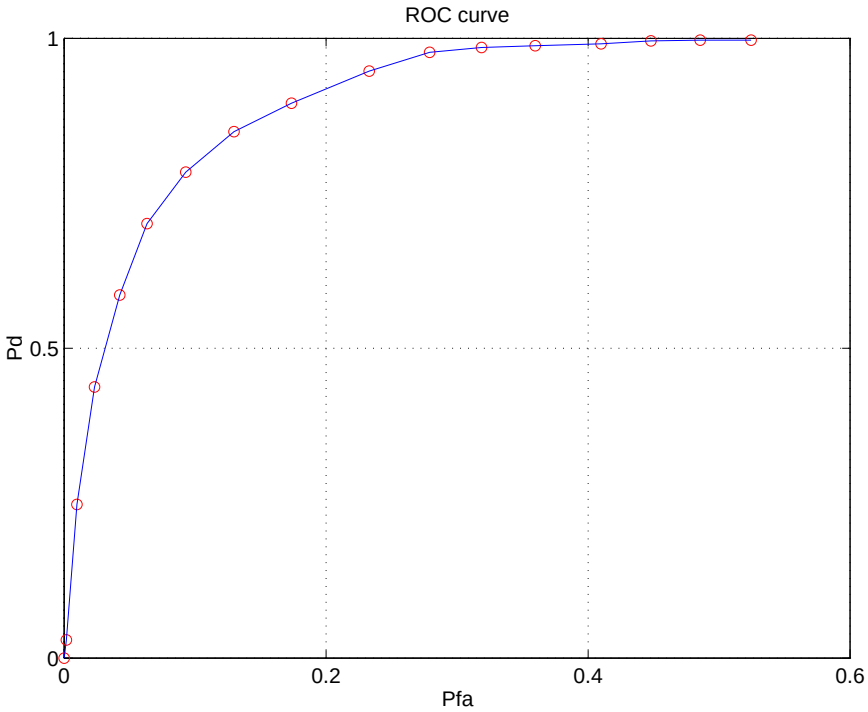

Figure 4.19: ROC curve for two-class problem.

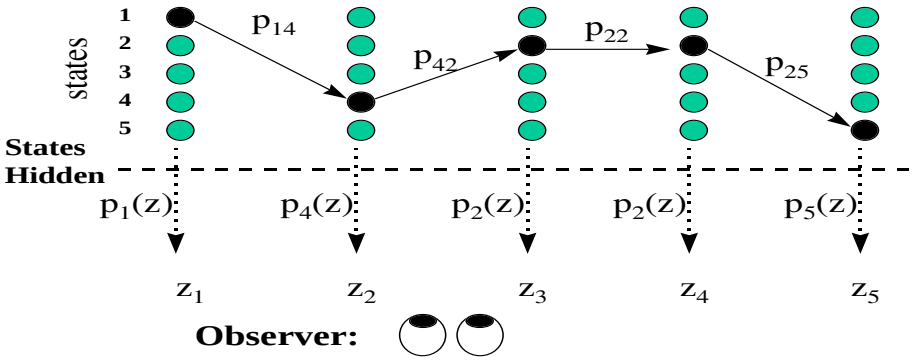

Figure 4.20: A hidden Markov model (HMM). As the state transitions occur from sample to sample, the observer, cannot see the states directly. Instead, the observer makes observations whose PDF depends on the state.

probability that the model is in state j at time t is governed only by the transition probability a_{ij}, where i is the true state at time $t-1$. The Markov model is "hidden" from view by the observer who can only observe measurements $\mathbf{z}_t$ whose PDF is governed by the true state at each time step. The mathematics of the HMM are reviewed in section 4.4.2.

How HMM's are used.

The Baum-Welch algorithm is an algorithm for estimating the parameters of the HMM from training data. The HMM is a complete statistical model for the series of measurements $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_T$ and therefore defines the probability density function $p(\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_T)$. Therefore, once the parameters have been determined, it is easy to use the HMM as a classifier. Furthermore, it is also easy to generate "typical" measurement sequences. This aspect of the HMM has always fascinated me since in principle, it would be possible to train an HMM on a specific human speaker, then create totally random "jibberish" that sounded like the same speaker. I have always wondered if certain politicians are already using such a device. For further information on HMM's, the reader is referred to the tutorial by Rabiner [41].

The role of HMM's in class-specific classifiers

In classifying signals, The hidden Markov model (HMM) has a major advantage but one serious drawback. The advantages is that complex processes may be modeled using low-dimensional models, thereby allowing the HMM to be trained using a realizable amount of data. The low dimension is achieved by dividing (segmenting) the data into small time steps from which low-dimensional measurements are made. Although the total observation space is large (the number of steps times the dimension of the observations), the dimension of the observations may be kept low.

But the problem with HMM's is that they need to be carefully tailored for a specific type of random process. Not only is the segment size chosen specially, but so is the observation space (the feature set). It is difficult for an HMM designed for speech recognition to operate well for other types of processes except speech. If separate HMM's are used, the likelihood values cannot be directly compared in a classifier. The class-specific method solves this problem by allowing two or more HMM's to be used as detectors for their respective model class, while solving the problem of comparing the outputs optimally.

4.4.2 The standard HMM

Following the notation of Rabiner [41], there are T observation times. At each time $1 \leq t \leq T$, there is a discrete state variable q_t which takes one of N values $q_t \in \{S_1, S_2, \cdots, S_N\}$. According to the Markovian assumption, the probability distribution of q_{t+1} depends only on the value of q_t. This is described compactly as a state transition probability matrix A whose elements a_{ij} represent the probability that q_{t+1} equals j given that q_t equals i. The initial state probabilities are denoted π_i, the probability that q_1 equals S_i.

It is a **hidden** Markov model because the states q_t are hidden from view; we cannot observe them. But, we can observe the random data O_t which is generated according to a PDF dependent on the state at time t. We denote the PDF of O_t under state j as $b_j(O_t)$.

The complete set of model parameters that define the HMM are

$$\Lambda = \{\pi_j, a_{ij}, b_j\}$$

The Baum-Welch algorithm calculates new estimates Λ given an observation sequence $\mathbf{O} = O_1 O_2 \cdots O_T$ and a previous estimate of Λ. The algorithm is composed of two parts: the forward/backward procedure, and the reestimation of parameters.

Using Gaussian Mixtures for $b_j(O)$.

It will be convenient to model the PDF's $b_j(O_t)$ as Gaussian mixtures:

$$b_j(O) = \sum_{m=1}^{M} c_{jm}\, \mathcal{N}(O, \boldsymbol{\mu}_{jm}, \mathbf{U}_{jm}), \quad 1 \leq j \leq N$$

where

$$\mathcal{N}(O, \boldsymbol{\mu}_{jm}, \mathbf{U}_{jm}) = (2\pi)^{-P/2} |\mathbf{U}_{jm}|^{-1/2} \exp\left\{ -\frac{1}{2}(O - \boldsymbol{\mu}_{jm})' \mathbf{U}_{jm}^{-1} (O - \boldsymbol{\mu}_{jm}) \right\},$$

and P is the dimension of O. We will refer to these Gaussian mixture parameters collectively as

$$b_j \triangleq \{c_{jm}, \boldsymbol{\mu}_{jm}, \mathbf{U}_{jm}\}.$$

Forward/Backward Procedure

We wish to compute the probability of observation sequence $\mathbf{O} = O_1 O_2 \cdots O_T$ given the model $\Lambda = \{\pi_j, a_{ij}, b_j\}$. The *forward procedure* for $p(\mathbf{O}|\Lambda)$ is

1. Initialization:
$$\alpha_1(i) = \pi_i\, b_i(O_1), \quad 1 \leq i \leq N \tag{4.9}$$
2. Induction:
$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij} \right] b_j(O_{t+1}), \qquad \begin{array}{l} 1 \leq t \leq T-1 \\ 1 \leq j \leq N \end{array} \tag{4.10}$$
3. Termination:
$$p(\mathbf{O}|\Lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{4.11}$$

The *backward procedure* is

1. Initialization:
$$\beta_T(i) = 1 \tag{4.12}$$
2. Induction:
$$\beta_t(i) = \sum_{j=1}^{N} a_{ij}\, b_j(O_{t+1})\, \beta_{t+1}(j), \qquad \begin{array}{l} t = T-1, T-2, \cdots, 1 \\ 1 \leq i \leq N \end{array} \tag{4.13}$$

Reestimation of HMM Parameters

The *reestimation* procedure calculates new estimates of Λ given the observation sequence $\mathbf{O} = O_1 O_2 \cdots O_T$. We first define

$$\xi_t(i,j) = \frac{\alpha_t(i)\ a_{ij}\ b_j(O_{t+1})\ \beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(i)\ a_{ij}\ b_j(O_{t+1})\beta_{t+1}(j)} \tag{4.14}$$

and

$$\gamma_t(i) = \sum_{j=1}^{N}\xi_t(i,j). \tag{4.15}$$

The updated state priors are

$$\hat{\pi}_i = \gamma_1(i). \tag{4.16}$$

The updated state transition matrix is

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1}\xi_t(i,j)}{\sum_{t=1}^{T-1}\gamma_t(i)}. \tag{4.17}$$

Reestimation of Observation PDF's

In order to update the observation PDF's, it is necessary to maximize

$$Q_j = \sum_{t=1}^{T} w_{tj}\log b_j(O_t).$$

over the PDF b_j, where

$$w_{t,j} = \frac{\alpha_t(j)\ \beta_t(j)}{\sum_{i=1}^{N}\alpha_t(i)\ \beta_t(i)}. \tag{4.18}$$

This is a weighted maximum likelihood (ML) procedure since if $w_{tj} = c_j$, the results are the strict ML estimates. The weights w_{tj} are interpreted as the probability that the Markov chain is in state j at time t.

Reestimation of Gaussian Mixture Parameters

If $b_j(O)$ are modeled as Gaussian mixtures (GM), one could simply determine the weighted ML estimates of the GM parameters. Since only iterative methods are known, this would require iterating to convergence at each step. A more global approach is possible if the mixture component assignments are regarded as "missing data" [42]. The result is that the quantity

$$Q_j = \sum_{t=1}^{T}\sum_{m=1}^{M}\gamma_t(j,m)\log b_j(O_t) \tag{4.19}$$

is maximized, where

$$\gamma_t(j,m) = w_{t,j}\left[\frac{c_{jm}\ \mathcal{N}(\mathbf{O}_t,\boldsymbol{\mu}_{jm},\mathbf{U}_{jm})}{\sum_{k=1}^{M}c_{jk}\ \mathcal{N}(\mathbf{O}_t,\boldsymbol{\mu}_{jk},\mathbf{U}_{jk})}\right] \tag{4.20}$$

The weights $\gamma_t(j,m)$ are interpreted as the probability that the Markov chain is in state j and the observation is from mixture component m at time t. The resulting update equations for c_{jm}, $\boldsymbol{\mu}_{jm}$, and $\mathbf{U}_{jm}$ are computed as follows:

$$\hat{c}_{jm} = \frac{\sum_{t=1}^{T} \gamma_t(j,m)}{\sum_{t=1}^{T}\sum_{l=1}^{M} \gamma_t(j,l)} \tag{4.21}$$

Note the similarity to (4.2). This means that the algorithms designed for Gaussian mixtures are applicable for updating the state PDFs of the HMM.

$$\hat{\boldsymbol{\mu}}_{jm} = \frac{\sum_{t=1}^{T} \gamma_t(j,m)\ \mathbf{O}_t}{\sum_{t=1}^{T} \gamma_t(j,m)} \tag{4.22}$$

$$\hat{\mathbf{U}}_{jm} = \frac{\sum_{t=1}^{T} \gamma_t(j,m)\ (\mathbf{O}_t - \boldsymbol{\mu}_{jm})\ (\mathbf{O}_t - \boldsymbol{\mu}_{jm})'}{\sum_{t=1}^{T} \gamma_t(j,m)} \tag{4.23}$$

Note that the above equations do not treat the problem of constraining the GM covariances. This needs to be addressed (see section 4.3).

Structured State Transition Matrices

TBD

Multiple Records

It is fairly straight-forward to extend the Baum-Welch algorithm to the case when multiple observation sequences ("records") are available. Rather than $O_1, O_2, \ldots, O_T$, we have $O_1^r, O_2^r, \ldots, O_{T_r}^r, \quad r = 1, 2, \ldots, R$. For each record,

1. Run the forward-backward procedure on $O_1^r, O_2^r, \ldots, O_{T_r}^r$ to produce $\alpha_t^r(i)$, $\beta_t^r(i)$,
2. Compute $\xi_t^r(i,j)$, $t = 1, \ldots, T_r$ as in (4.14).
3. Compute $\gamma_t^r(i)$ as in (4.15).

Then, we have

$$\hat{\pi}_i = \sum_{r=1}^{R} \gamma_1^r(i) \qquad \hat{a}_{ij} = \frac{\sum_{r=1}^{R}\sum_{t=1}^{T_r-1} \xi_t^r(i,j)}{\sum_{r=1}^{R}\sum_{t=1}^{T_r-1} \gamma_t^r(i)}.$$

Updating the Gaussian mixture parameters requires defining

$$w_{t,j}^r = \frac{\alpha_t^r(j)\ \beta_t^r(j)}{\sum_{i=1}^{N} \alpha_t^r(i)\ \beta_t^r(i)},$$

which leads to $\gamma_t^r(j,k)$ through (4.20). We then have

$$\hat{c}_{jm} = \frac{\sum_{r=1}^{R}\sum_{t=1}^{T_r}\gamma_t^r(j,m)}{\sum_{r=1}^{R}\sum_{t=1}^{T_r}\sum_{l=1}^{M}\gamma_t^r(j,l)}$$

and

$$\hat{\boldsymbol{\mu}}_{jm} = \frac{\sum_{r=1}^{R}\sum_{t=1}^{T_r}\gamma_t^r(j,m)\ \mathbf{O}_t}{\sum_{r=1}^{R}\sum_{t=1}^{T_r}\gamma_t^r(j,m),}$$

... et cetera.

4.4.3 MATLAB toolbox for HMM

We will demonstrate the HMM toolbox by example.

An HMM example

We now describe a simple problem that we will analyze using the HMM tools. Consider the HMM with the following parameters:

$$A = \begin{bmatrix} .8 & .1 & .1 \\ .1 & .8 & .1 \\ .1 & .1 & .8 \end{bmatrix} \quad \pi = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

The output of the HMM is a time series with a 16-sample step size (i.e. the state is allowed to change every 16 output samples). The output is Gaussian with mean and variance depending on the state as follows:

State	Mean	Var
1	0	1
2	0	4
3	2	1

For each 16-sample segment, the sample mean and standard deviation are computed. This constitutes a 2-dimensional feature vector that is the observation space of the HMM.

Creating feature data for training.

To test the tools, we need to generate HMM output data from the above-defined model. Execute the script file `hmm_example.m`. The program calls the function `hmm_maketestdata.m` which generates the 2-dimensional feature data as described above. The call is

```
[x,istart,nsamp]=hmm_maketestdata(Pi, A,nre cord, nsteps ,N,NF EAT);
```

There are 10 records of length 400 segments, thus `x` is size 2-by-4000. The auxiliary outputs `istart,nsamp` are vectors containing the starting samples and lengths of each of the ten records. This makes it possible to locate individual records within the matrix. The script then plots the data using the command

```
plot(x(1,:),x(2,:),'b.');
xlabel('MEAN');
ylabel('STDV');
```

and waits for keyboard input. The resulting figure is shown in Figure 4.21.

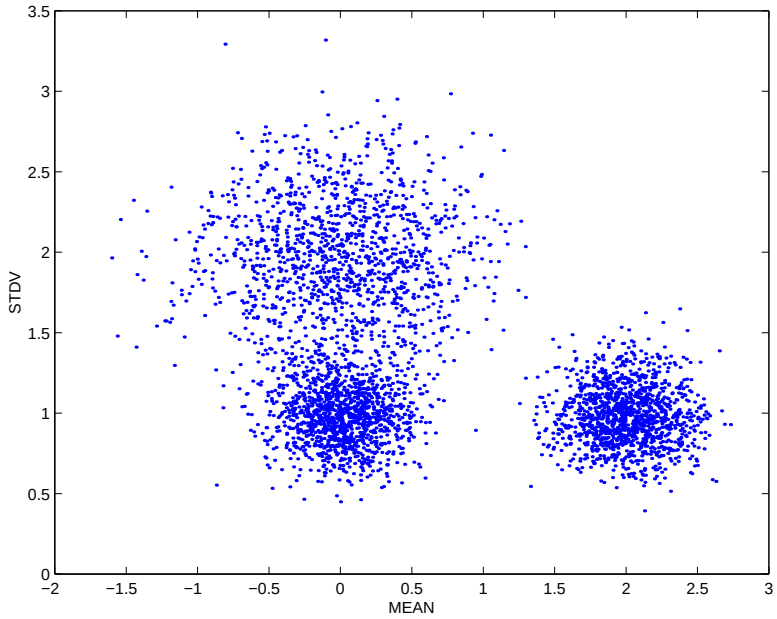

Figure 4.21: Scatter plot of the HMM output features. The three states can be seen individually. Compare the plot with the table of means and standard deviations.

Initializing HMM parameters

Next, initialize a set of HMM parameters using the commands.

```
names={'MEAN','STDV'};
min_std=[.1    .1];
NSTATES=3;
NMODE=10;
parm=init_hmm(x,NSTATES,NMODE,names,mi          n_std );
```

This first two commands define the feature names and the minimum standard deviations for Gaussian mixture estimation (See Section 4.3). The initial HMM parameters are obtained by using `init_hmm.m` which creates a uniform state transition matrix A and prior probability π. The PDF of the feature vector in each state is approximated by Gaussian mixtures. The starting point for the Gaussian mixture parameters are obtained by the function `init_gmix.m` described in the previous sections.

Training using the Baum-Welch algorithm

To run 10 iterations of the Baum-Welch algorithm, use the commands:

```
NIT=100;
[q,  parm]  = hmm_reest(parm,     x,  istart,   nsamp,   NIT);
```

The algorithm prints the total log likelihood at each iteration. At the end, it prints the final state transition matrix and initial probabilities. These should be close to the correct ones.

Viewing the state PDF's

To view the HMM PDF's, execute the command

```
hmm_view(parm,x,1,2);
```

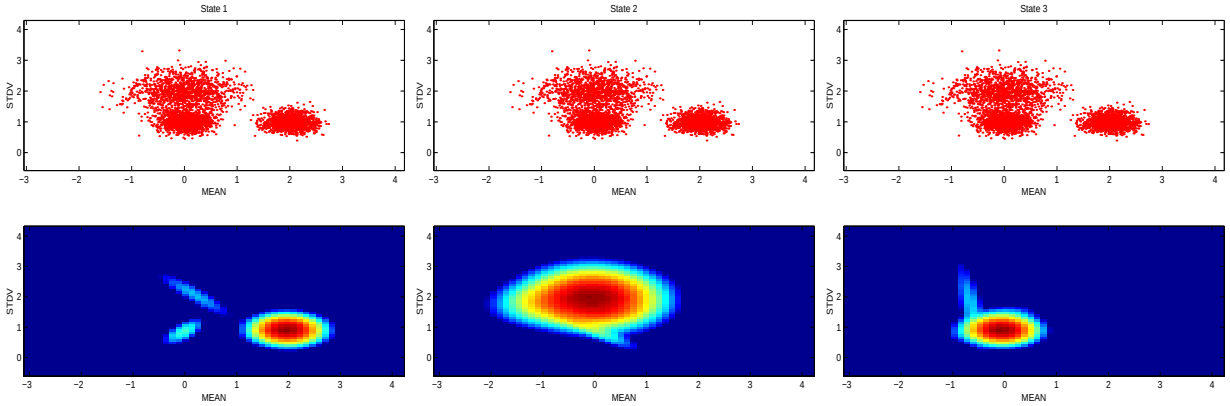

Figure 4.22: PDF plots of the three state PDF's after convergence. Aside from some minor outlier modes, the PDF estimates correctly approximate the true PDF's. It is easy to see which PDF corresponds to which state of the simulated HMM.

This produces the three state PDF plots as shown in Figure 4.22. The last two arguments are the indexes of the two dimensions to be viewed. Since there are only two dimensions, the only choice is `1,2` (See a description of `gmix_view2.m` in Section 4.3). Look at the figure and try to figure out which PDF corresponds to state 1, 2, and 3. If a bad starting point was used, it may not have worked.

Annealing

No matter how many iterations one makes, the bad solution will never converge to the correct. But there is a method that is usually successful in nudging a solution away from a bad stationary point. This we call *annealing* and is done by expanding the covariance matrices of the PDF estimates and by pushing the state transition matrix and prior state probabilities closer to "uniform". The utility `ann_hmm` does this. Attempt to find a "bad" stationary point by re-running the above sequence until one is found. Next, use the commands

```
parm=ann_hmm(parm,2,1.2);
[log_pdf_val,  parm]  = hmm_reest(parm,  x,  istart,  nsamp,  NIT);
hmm_view(parm,x,1,2);
```

This should correct the problem. Try it to satisfy yourself that it works. The second argument is the expansion factor for Cholesky factors of the covariance matrices and the third is a parameters greater than 1.0 that determines how much the state transition matrix is annealed.

Creating Synthetic Observations

Creating sequences of observations corresponding to an HMM parameter set is simple. The command

```
[x2,states]=hmm_synth_mex(parm,100);
x2=x2';
```

creates a record of 100 observations from the HMM defined by parameter set `parm` . The output vector is of "nsamp" rows and number of columns corresponding to the feature dimension. It has to be transposed to agree with the normal convention. The states are passed to the output as variable "states".

Estimating the states: the Viterbi algorithm.

The Viterbi algorithm [41] estimates the most likely state sequence. The command:

```
states=viterbi(parm,x);
```

Outputs the most likely state sequence corresponding to data `x`. As a test, try the following commands:

```
[x2,states]=hmm_synth_mex(parm,100);
x2=x2';
est_states=viterbi(parm,x2);
```

Compare the estimated states with the actual.

Classifying using the trained HMM parameters

The log-likelihood output of the `train_hmm` program can be used as a classifier. If the number of iterations is specified as zero, a shortened version of the program is run, only running the forward procedure.

```
[q, parm] = hmm_reest(parm, x, istart, nsamp, 0);
```

Since the program finds the total log likelihood for each record passed to it, the total likelihood will be the sum of the elements of `q`.

Bibliography

[1] Duda and Hart, *Pattern Classification and Scene Analysis*. Wiley, 1973.

[2] A. Webb, *Statistical Pattern Recognition*. London: Arnold, 1999.

[3] J. Schurmann, *Statistical Pattern Recognition*. New York: Wiley, 1996.

[4] R. E. Bellman, *Adaptive Control Processes*. Priceton, New Jersey, USA: Princeton Univ. Press, 1961.

[5] C. J. Stone, "Optimal rates of convergence for nonparametric estimators," *Annals of Statistics*, vol. 8, no. 6, pp. 1348–1360, 1980.

[6] D. W. Scott, *Multivariate Density Estimation*. Wiley, 1992.

[7] S. Aeberhard, D. Coomans, and O. de Vel, "Comparative analysis of statistical pattern recognition methods in high dimensional settings," *Pattern Recognition*, vol. 27, no. 8, pp. 1065–1077, 1994.

[8] S. J. Raudys and A. K. Jain, "Small sample size effects in statistical pattern recognition: Recommendations for practitioners," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 13, no. 3, pp. 252–264, 1991.

[9] N. Intrator, *Feature Extraction Using an Exploratory Projection Pursuit Neural Network*. PhD thesis, Brown University, 1991.

[10] P. J. Huber, "Projection pursuit," *Annals of Statistics*, vol. 13, no. 2, pp. 435–475, 1985.

[11] P. M. Baggenstoss, "Structural learning for classification of high dimensional data," in *Proceedings of the 1997 International Conference on Intelligent Systems and Semiotics*, pp. 124–129, National Institute of Standards and Technology, 1997.

[12] A. Finch, "A neural network for dimension reduction and application to image segmentation," in *Proceedings of the 1994 International Conference on Artificial Neural Networks (ICANN-94)*, pp. 252–264, 1994.

[13] H. Watanabe, *Knowing and Guessing*. New York: John Wiley, 1969.

[14] T. Kohonen, G. Németh, K.-J. Bry, M. Jalanko, and H. Riittinen, "Spectral classification of phonemes by learning subspaces," in *Proc. ICASSP 79*, pp. 97–100, 1979.

[15] E. Oja, *Subspace Methods of Pattern Recognition*. Research Studies Press, 1983.

[16] B. Schoelkopf, C. Burges, and V. N. Vapnik, "Extracting support data for a given task," in *Proc. 1st Int. Conf. Knowledge Discovery Data Mining* (U.M.Fayyad and R. Uthurusamy, eds.), (Menlo Park, CA), AAAI Press, 1995.

[17] K. Mueller, S. Mika, G. Raetsch, K. Tsuda, and B. Schoelkopf, "An introduction to kernel-based learning algorithms," *IEEE Trans. Neural Networks*, vol. 12, no. 2, pp. 181–201, 2001.

[18] Frimpong-Ansah, K. Pearce, D. Holmes, and W. Dixon, "A stochastic/feature based recogniser and its training algorithm," *ICASSP-89*, vol. 1, pp. 401–404, 1989.

[19] S. Kumar, J. Ghosh, and M. Crawford, "A versatile framework for labeling imagery with large number of classes," in *Proceedings of the International Joint Conference on Neural Networks*, (Washington, D.C.), pp. 2829–2833, 1999.

[20] S. Kumar, J. Ghosh, and M. Crawford, "A hierarchical multiclassifier system for hyperspectral data analysis," in *Multiple Classifier Systems* (J. Kittler and F. Roli, eds.), pp. 270–279, Springer, 2000.

[21] H. Watanabe, T. Yamaguchi, and S. Katagiri, "Discriminative metric design for robust pattern recognition," *IEEE Trans. Signal Processing*, vol. 45, no. 11, pp. 2655–2661, 1997.

[22] P. Belhumeur, J. Hespanha, and D. Kriegman, "Eigenfaces vs. Fisherfaces: Recognition using class specific linear projection," *PAMI*, vol. 19, pp. 711–720, July 1997.

[23] D. Sebald, "Support vector machines and the multiple hypothesis test problem," *IEEE Trans. Signal Processing*, vol. 49, pp. 2865–2872, November 2001.

[24] I.-S. Oh, J.-S. Lee, and C. Y. Suen, "A class-modularity for character recognition," in *Proceedings of International Conference on Document Analysis and Recognition (ICDAR) 2001*, (Seattle, Washington), pp. 64–68, September 2001.

[25] E. Sali and S. Ullman, "Combining class-specific fragments for object classification," in *Proceedings of 1999 British Machine Vision Conference (BMVC99)*, (University of Nottingham), pp. 203–213, September 1999.

[26] D. A. Landgrebe, S. B. Serpico, M. M. Crawford, and V. Singhroy, "Introduction to the special issue on analysis of hyperspectral image data," *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 39, pp. 1343 – 1345, July 2001.

[27] S. M. Kay, A. H. Nuttall, and P. M. Baggenstoss, "Multidimensional probability density function approximation for detection, classification and model order selection," *IEEE Trans. Signal Processing*, pp. 2240–2252, Oct 2001.

[28] H. W. Sorensen, *Parameter Estimation, Principles ans Problems*. New York: Marcel Dekker, 1980.

[29] D. R. Cox and D. V. Hinkley, *Theoretical Statistics*. London: Chapman and Hall, 1974.

[30] R. L. Strawderman, "Higher-order asymptotic approximation: Laplace, saddlepoint, and related methods," *Journal of the American Statistical Association*, vol. 95, pp. 1358–1364, December 2000.

[31] J. Durbin, "Approximations for densities of sufficient estimators," *Biometrika*, vol. 67, no. 2, pp. 311–333, 1980.

[32] H. L. Royden, *Real Analysis*. Englewood Cliffs, New Jersey, USA: Prentice Hall, third ed., 1988.

[33] T. Minka, "Exemplar-based likelihoods using the pdf projection theorem.," *Microsoft Research Ltd, technical report*, March 2004.

[34] C. Bell, H. Fujisaki, J. Heinz, K. Stevens, and A. House, "Reduction of speech spectra by analysis-by-synthesis techniques," *Journal of the Acoustical Society of America*, pp. 1725–1736, December 1961.

[35] E. Parzen, "On estimation of a probability density function and mode," *Annals of Mathematical Statistics*, vol. 33, pp. 1065–1076, 1962.

[36] D. M. Titterington, A. F. M. Smith, and U. E. Makov, *Statistical Analysis Of Finite Mixture Distributions*. John Wiley & Sons, 1985.

[37] R. A. Redner and H. F. Walker, "Mixture densities maximum likelihood, and the EM algorithm," *SIAM Review*, vol. 26, April 1984.

[38] N. Vlassis and A. Likas, "The kurtosis-EM algorithm for Gaussian mixture modelling," *IEEE Trans. SMC (submitted)*, 1999.

[39] Anderson and Moore, *Optimal Filtering*. PH, 1979.

[40] M. Kendall and A. Stuart, *The Advanced Theory of Statistics, Vol. 2*. London: Charles Griffin, 1979.

[41] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, pp. 257–286, February 1989.

[42] B. H. Juang, "Maximum likelihood estimation for mixture multivariate stochastic observations of Markov chains," *AT&T Technical Journal*, vol. 64, no. 6, pp. 1235–1249, 1985.

[43] A. Oppenheim and R. Schafer, "Homomorphic analysis of speech," *IEEE Trans. Audio Electroacoustics*, vol. AU-16, pp. 221–226, 1968.

[44] J. W. Picone, "Signal modeling techniques in speech recognition," *Proceedings of the IEEE*, vol. 81, no. 9, pp. 1215–1247, 1993.

[45] S. Kay, *Modern Spectral Estimation: Theory and Applications*. Prentice Hall, 1988.

[46] U. Viswanathan and J. Makhoul, "Quantization properties of transmission parameters in linear predictive systems," *IEEE Trans. ASSP*, vol. 23, pp. 309–321, 1975.

[47] O. E. Barndorff-Nielsen and D. R. Cox, *Asymptotic Techniques for Use in Statistics*. Chapman and Hall, 1989.

[48] A. H. Nuttall, "Saddlepoint approximation and first-order correction term to the joint probability density function of M quadratic and linear forms in K Gaussian random variables with arbitrary means and covariances," *NUWC Technical Report 11262*, December 2000.

[49] A. H. Nuttall, "Joint probability density function of selected order statistics and the sum of the remaining random variables," *NUWC Technical Report 11345*, October 2001.

[50] A. H. Nuttall, "Joint probability density function of selected order statistics and the sum of the remainder as applied to arbitrary independent random variables," *NUWC Technical Report 11469*, November 2003.

[51] A. H. Nuttall, "Saddlepoint approximation for the combined probability and joint probability density function of selected order statistics and the sum of the remainder," *NUWC Technical Report 11XXX*, February 2004.

[52] L. I. Perlovsky, "Efficient neural network for transient signal classification," *1990 Asilmar Conference on Signals, Systems and Computing*, 1990.

[53] T. E. Luginbuhl, *Estimation of General Discrete-Time Frequency Modulated Processes*. PhD thesis, University of Connecticut, 1999.

[54] A. H. Nuttall, "Detection performance of generalized likelihood ratio processors for random signals of unknown location, structure, extent, and strength," *NUWC Technical Report 10739*, August 1994.